GREIS

GNSS Receiver External Interface Specification

Reflects Firmware Version 3.0.0

Last revised: February 27, 2009

Table of Contents

PREFACE

Thank you for purchasing your JAVAD GNSS receiver. The materials available in this Reference Guide (the "Guide") have been prepared by JAVAD GNSS, Inc. for owners of JAVAD GNSS products. It is designed to assist owners with the use of the receiver and its use is subject to these terms and conditions (the "Terms and Conditions").

Terms and Conditions

PROFESSIONAL USE – JAVAD GNSS receivers are designed to be used by a professional. The user is expected to have a good knowledge and understanding of the user and safety instructions before operating, inspecting or adjusting. Always wear the required protectors (safety shoes, helmet, etc.) when operating the receiver.

DISCLAIMER OF WARRANTY – EXCEPT FOR ANY WARRANTIES IN THIS GUIDE OR A WARRANTY CARD ACCOMPANYING THE PRODUCT, THIS GUIDE AND THE RECEIVER ARE PROVIDED "AS-IS." THERE ARE NO OTHER WARRANTIES. JAVAD GNSS DISCLAIMS ANY IMPLIED WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR USE OR PURPOSE. JAVAD GNSS AND ITS DISTRIBUTORS SHALL NOT BE LIABLE FOR TECHNICAL OR EDITORIAL ERRORS OR OMISSIONS CONTAINED HEREIN; NOR FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES RESULTING FROM THE FURNISHING, PERFORMANCE OR USE OF THIS MATERIAL OR THE RECEIVER. SUCH DISCLAIMED DAMAGES INCLUDE BUT ARE NOT LIMITED TO LOSS OF TIME, LOSS OR DESTRUCTION OF DATA, LOSS OF PROFIT, SAVINGS OR REVENUE, OR LOSS OF THE PRODUCT'S USE. IN ADDITION, JAVAD GNSS IS NOT RESPONSIBLE OR LIABLE FOR DAMAGES OR COSTS INCURRED IN CONNECTION WITH OBTAINING SUBSTITUTE PRODUCTS OR SOFTWARE, CLAIMS BY OTHERS, INCONVENIENCE, OR ANY OTHER COSTS. IN ANY EVENT, JAVAD GNSS SHALL HAVE NO LIABILITY FOR DAMAGES OR OTHERWISE TO YOU OR ANY OTHER PERSON OR ENTITY IN EXCESS OF THE PURCHASE PRICE FOR THE RECEIVER.

LICENSE AGREEMENT – Use of any computer programs or software supplied by JAVAD GNSS or downloaded from a JAVAD GNSS website (the "Software") in connection with the receiver constitutes acceptance of these Terms and Conditions in this Guide and an agreement to abide by these Terms and Conditions. The user is granted a personal, non-exclusive, non-transferable license to use such Software under the terms

stated herein and in any case only with a single receiver or single computer. You may not assign or transfer the Software or this license without the express written consent of JAVAD GNSS. This license is effective until terminated. You may terminate the license at any time by destroying the Software and Guide. JAVAD GNSS may terminate the license if you fail to comply with any of the Terms or Conditions. You agree to destroy the Software and Guide upon termination of your use of the receiver. All ownership, copyright and other intellectual property rights in and to the Software belong to JAVAD GNSS. If these license terms are not acceptable, return any unused software and guide.

CONFIDENTIALITY – This guide, its contents and the Software (collectively, the "Confidential Information") are the confidential and proprietary information of JAVAD GNSS. You agree to treat JAVAD GNSS' Confidential Information with a degree of care no less stringent that the degree of care you would use in safeguarding your own most valuable trade secrets. Nothing in this paragraph shall restrict you from disclosing Confidential Information to your employees as may be necessary or appropriate to operate or care for the receiver. Such employees must also keep the Confidentiality Information confidential. In the event you become legally compelled to disclose any of the Confidential Information, you shall give JAVAD GNSS immediate notice so that it may seek a protective order or other appropriate remedy.

WEBSITE; OTHER STATEMENTS – No statement contained at the JAVAD GNSS website (or any other website) or in any other advertisements or JAVAD GNSS literature or made by an employee or independent contractor of JAVAD GNSS modifies these Terms and Conditions (including the Software license, warranty and limitation of liability).

SAFETY – Improper use of the receiver can lead to injury to persons or property and/or malfunction of the product. The receiver should only be repaired by authorized JAVAD GNSS warranty service centers.

MISCELLANEOUS – The above Terms and Conditions may be amended, modified, superseded, or canceled, at any time by JAVAD GNSS. The above Terms and Conditions will be governed by, and construed in accordance with, the laws of the State of California, without reference to conflict of laws.

Introduction

1.1 What is GREIS

GREIS is an interfacing language enabling the user to effectively communicate with GNSS receivers by accessing all of their capabilities and functions.

GREIS represents a generic receiver language structure for the entire range of JAVAD GNSS hardware. This language structure is receiver-independent and open to future modification or expansion. GREIS is based on a unified approach allowing the user to control a JAVAD GNSS receiver using an appropriate set of *named objects*. Communication with these objects is achieved through predefined *commands* and *messages*. There are no specific constraints on the number or type of the receiver objects used.

1.2 How is GREIS Used

Any system communicating with the JAVAD GNSS receiver through one of its ports (serial, parallel, USB, Ethernet, etc.) will use GREIS commands and messages to accomplish the required task. A pair of typical applications where GREIS plays a very important role are, first, using hand-held controllers to communicate with the receivers during field operation in survey and RTK projects or, second, when downloading data from the receivers into desktop systems for further post processing. A post processing application itself doesn't use GREIS commands, but needs to be aware of GREIS messages to extract data from the data files.

One important feature of GREIS is that it can be effectively used both for the automatic and manual control of JAVAD GNSS receivers. For manual control, the user will enter necessary GREIS commands into the receiver through a terminal. This is easily achievable as GREIS is designed to be human-readable text interface. On the other hand, GREIS obeys rather strict rules that makes it easy to use by applications.

1.3 Lists

GREIS heavily utilizes a concept of *lists*. Lists are used both in the receiver input language and in the standard text messages.

Lists in GREIS are represented by a sequence of elements delimited with commas (`,`, ASCII code 44), and enclosed into braces (`{}`, ASCII codes 123 and 125):

```
{element1,element2,element3}
```

In turn, elements of a list may themselves be lists:

```
{e1,{ee21,ee22},e3}
```

Note that the above definition is recursive, so that lists of arbitrary nesting depth are allowed. Elements that are not lists are called *leaf elements*, or simply *leafs*. Elements of lists could be empty, in which case we say the element is *omitted*. For example, in the list below, second element is omitted:

```
{e1,,e3}
```

Spaces before and after delimiters are allowed and ignored.

If elements of a list all have the same substring (prefix) at the beginning, this substring could be moved out of the braces surrounding the list, e.g.,

```
elem{1,2,3}
```

is a shorter form of the

```
{elem1,elem2,elem3}
```

Elements could be enclosed into double-quotes (`"`, ASCII code 34) that are stripped during parsing. Inside quoted element, special symbols (braces, commas, etc.) loose their role and are considered to be regular characters. Another use of quotes is to distinguish between “element is not specified” and “empty element specified” conditions. The former is denoted by simply omitting an element from the list, and the latter is denoted by putting pair of double-quotes between the commas. Quoting is also useful when one needs to have leading or trailing spaces in a string.

To put double-quote into element, quote this element and escape the double-quote inside with the backslash character (`\`, ASCII code 92). To put backslash by itself into quoted string, escape it with another backslash, for example:

Example:

```
"String with \"quotes\", backslash \\, and special characters, {}"
```

1.4 Objects

In the context of the model that GREIS is based on, a JAVAD GNSS receiver is identified with a set of *named objects*.

Object is defined as a hardware or software entity of the receiver's that can be addressed, set, or queried. Hardware entities are commonly referred to as *devices*, whereas firmware objects are normally *files* and *parameters*. Receiver ports and memory modules are all good examples of devices. All devices, files and parameters are treated in a uniform way by GREIS. Every object has an associated set of attributes that can be accessed, defined, and/or changed through GREIS.

1.4.1 Object Identifiers

It has been already mentioned that a receiver is considered as a set of objects (devices, files, messages, parameters, etc.) in the context of the GREIS model. For the purposes of addressing the objects in the receiver commands, a unique identifier should be assigned to every object.

Objects in the receiver are logically organized into groups. A group itself is also an object and belongs to another group unless it is the root group. Thus all objects in the receiver are organized into a tree-like hierarchy starting at the single root group. This representation resembles the organization of files into directories (folders) that most computer users are familiar with.

In GREIS, object groups are represented as *lists* of corresponding object names. The object name is unique inside the list to which the object belongs. Globally unique object identifier is defined as all the object names on the path through the object tree from the root list to the object, delimited by the forward slash (/). The root list itself is identified by the single forward slash.

Examples of object identifiers are:

Example: The root group:

```
/
```

Example: Receiver electronic ID:

```
/par/rcv/id
```

Example: Serial Port A baud rate:

```
/par/dev/ser/a/rate
```

Example: Attributes (size and last modification time) of the file `NAME` (file attributes are different from object attributes discussed below):

```
/log/NAME
```

Example: NMEA GGA sentence:

```
/msg/nmea/GGA
```

All the objects have one or more attributes associated with them. Object attributes are identified by appending the `&` character and the attribute name to the object identifier. The primary attribute each object has is `value`. This attribute is always accessed implicitly by GREIS commands. Some of objects may have additional attributes, for example:

Example: Serial port A default baud rate:

```
/par/dev/ser/a/rate&def
```

Example: Contents of the file `NAME`:

```
/log/NAME&content
```

Example: Number of files:

```
/log&count
```

1.4.2 Object Types

Every object in the receiver has GREIS type associated with it. The type of an object defines its behavior with respect to GREIS commands. Specifically, the type defines which values the object can take and which particular commands are applicable to the object.

Refer to "Primary Object Types" on page 57 for detailed description of currently supported object types.

1.5 Periodic Output

An important role in the receiver operation plays its ability to periodically output some information, such as different kinds of measurements, calculated values, etc., according to specified schedule. GREIS defines a rich set of *messages* containing different types of information in different formats that are minimal units of output, and provides methods to request *periodic output* of any combination of the messages in any order to any of the supported media suitable for data output. Any supported medium suitable for data output is called *output stream* in GREIS.

For every output stream, receiver maintains a list of messages that are currently enabled to be output to the stream, called *output list*. The order in which messages are output, matches the order of messages in the output list. In addition, every message that is

present in an output list has its own set of *scheduling parameters* associated with it. Scheduling parameters attached to a message in an output list define the schedule of output of this particular message into this particular output stream. GREIS provides three commands, `em`, `out`, and `dm`, to allow for efficient manipulation of output lists and scheduling parameters.

Message scheduling parameters comprise four fields: `period`, `phase`, `count`, and `flags`, each of which plays different role in the output schedule definition. Below we will describe how exactly their values affect the output, but basically, the `period` specifies interval between outputs of the message; `phase` specifies time shift of the moments of output with respect to time moments when current time is multiple of `period`; the `count`, when greater than zero, limits the number of times the message will be output; whereas `flags` filed allows for some fine tuning of the output process.

1.5.1 Output Period and Phase

The `period` and `phase` fields of the message scheduling parameters are floating point values in the range [0…86400) seconds. Their exact meaning is described below.

Note: When the `F_CHANGE` bit is set in the `flags` field of the scheduling parameters, the `phase` field looses its usual role and becomes "forced output period" instead. See description of the `F_CHANGE` flag below for details.

The receiver has its internal time grid that is defined by the *receiver clock* and the value of the `/par/raw/curmsint` parameter that defines the *step* of receiver *internal epochs*. Receiver internal epochs occur when *receiver time* is multiple of the *step*. In turn, receiver time is defined as the value of receiver clock modulo one day (86400 seconds). Receiver scans the output lists only at internal receiver epochs, so that no output could be generated more frequently than that.

Taking into account the internal time grid, the *period* and *phase* variables define the time moments of the output of a message as follows: receiver will output the message only at the receiver times T_{out} simultaneously satisfying the following two equations:

$$\begin{cases} T_{out} (\text{mod } \ period) = phase & (1) \\ T_{out} = N \cdot step & (2) \end{cases}$$

where N is integer number taking the values `[0,1,2,…,(86400/step)-1]`.

The first equation defines the basic rule of messages output, and the second one imposes additional constraints related to the internal receiver epochs. Note that in the most usual

case, when both *period* and *phase* are multiples of *step*, the second equation is satisfied automatically whenever the first equation is satisfied. Also note that if

$$86400 \ (\mathrm{mod}\ period) \neq 0,$$

the actual interval between the last message sent before the day rollover and the first message after the day rollover will be different from the value of *period*.

Consider a couple of examples illustrating this mechanism:

Example: Suppose *period* is 10s, *phase* is 2.2s, and *step* is 0.2s. As T_{out}, according to the second equation, can take only values that are multiple of *step*, the left part of the first equation will take the following values: 0, 0.2, 0.4, …, 9.8, 0, …, from which only value 2.2 matches *phase*. These matches will occur, and the message will be output, every time T_{out} takes one of the following values: 2.2s, 12.2s, 22.2s, etc.

Example: Suppose *period* is 10s, *phase* is 2.2s, and *step* is 0.5s. The receiver will not output the message since the above pair of simultaneous equations is never satisfied.

Example: Suppose *phase* > *period*. The receiver won't output the message at all as the first equation will never be satisfied.

1.5.2 Output Count

The `count` field of the message scheduling parameters is an integer value in the range [-256…32767) and serves two different purposes:

1. When the `count` is 0, unlimited number of messages will be output. When the `count` is greater than 0, it defines how many times the message will be output. In this case the counter is decremented by 1 every time the message is output, and when it becomes 0, the `F_DISABLED` bit is set in the flags field. The message scheduler doesn't output messages with `F_DISABLED` bit set.
2. When the `count` is set to a value in the range [-256…-1], the output of the message is not suppressed, and the `count` field serves entirely different purpose. It enables wrapping of the message into special [>>] message before output (see "[>>] Wrapper" on page 322). The value of `count` then used to set `id` field in the generated [>>] message.

Note: The wrapping feature is useful, for example, for a server application that gets messages from receiver and forwards them to multiple clients. It can request wrapping of arbitrary messages into the [>>] messages with different identifiers, unwrap the received messages, and dispatch the data to particular client(s) based on the received `id`. Utilizing this feature, such an application doesn't

need to be aware of any other data formats but the format of the [>>] message, and can use single channel of communication with the receiver to get and dispatch messages in different formats.

1.5.3 Output Flags

The `flags` field of the message scheduling parameters is a 16-bit wide bitfield. Each bit of this bitfield is a separate flag and serves different purpose. The following is a list of the message scheduling flags.

Table 1-1. Message Scheduling Flags

Bit#	HEX	Name
0	0x0001	F_OUT
1	0x0002	F_CHANGE
2	0x0004	F_OUT_ON_ADD
3	0x0008	F_NOTENA
4	0x0010	F_FIX_PERIOD
5	0x0020	F_FIX_PHASE
6	0x0040	F_FIX_COUNT
7	0x0080	F_FIX_FLAGS
8	0x0100	reserved
9	0x0200	reserved
10	0x0400	reserved
11	0x0800	F_DISABLED
12–15	0xF000	reserved

Note: Field names are introduced here only for the purpose of referring to them in this manual. There is no way to use them in the GREIS commands.

`F_OUT` – If this flag is set, the first messages after invocation of the corresponding command will be output at the internal receiver epoch closest to the command execution time no matter what is specified by the `period` scheduling parameter.

`F_CHANGE` – If this flag is set, the corresponding message will be output only if the message data have changed since the last output of the message to the given output stream. Receiver checks whether the message data have changed only at the moments defined by the equations (1),(2) where *phase* variable is set to zero, and *period* variable is set to the value of `period` field. The message scheduling parameter `phase`, which loses its original function in this case, now plays the role of a *forced output period.* “Forced output” means that the corresponding message will be output whether its contents will have changed or not at the time moments

defined by the equations (1),(2) where *period* variable is set to the value of the `phase` field, and *phase* variable is set to zero. If the field `phase` is zero, then the receiver performs no forced output so that the corresponding message will be output only on condition that its data have changed.

`F_OUT_ON_ADD` – If this flag is set, then the first message will be output immediately after executing the corresponding `em` or `out` command. This flag is ignored for majority of messages[1].

`F_NOTENA` – If this flag is set for a message in an output list, the `F_DISABLED` flag for this message won't be cleared when the message is enabled, and therefore its output will remain suspended. For example, this flag is used in order not to output some of the messages from the default set of messages when the user changes output period on the fly, without first disabling the output.

`F_FIX_PERIOD`, `F_FIX_PHASE`, `F_FIX_COUNT`, `F_FIX_PERIOD` – Being set to 1 in a scheduling parameters, prevent changes to corresponding field(s) of this scheduling parameters through `em` and `out` commands.

`F_DISABLED` – Is not explicitly programmable by the user. When one enables a message with a positive `count`, then, after this message has been output `count` times, the message scheduler sets this flag to 1. This flag is cleared to 0 when the message is re-enabled, unless `F_NOTENA` flag is set for this message.

1. Currently only two GREIS messages, [JP] and [MF], honour this flag.

Receiver Input Language

This chapter describes the syntax and semantics of the receiver input language. We begin with some examples to give the reader a feeling of the language, then turn to detailed syntax definition, and then describe all the defined commands along with their semantics.

2.1 Language Examples

Here are a few examples of real statements receiver understands along with receiver replies. You will find more examples of using particular commands in corresponding subsections. The input to the receiver is marked with the ⇒ character, while receiver output is marked with the ⇐ character:

Example: Ask receiver to print its electronic ID. Receiver generates the reply message shown:

```
⇒ print,/par/rcv/id<CR>
⇐ RE00C QP01234TR45<CR><LF>
```

Example: Ask receiver to set the baud rate of its serial port A to 9600. Receiver successfully executes the command and doesn't generate any reply.

```
⇒ set,/par/dev/ser/a/rate,9600<LF>
```

Example: Use the same command as in the previous example, but force receiver to generate reply by means of using the *statement identifier*.

```
⇒ %set_rate%set,/par/dev/ser/a/rate,9600<LF>
⇐ RE00A%set_rate%<CR><LF>
```

Example: Try to set too high baud rate. Receiver replies with the error message even though we used no statement identifier.

```
⇒ set,/par/dev/ser/a/rate,1000000<LF>
⇐ ER016{4,value out of range}<CR><LF>
```

Note: Receiver always puts its normal and error replies into two standard messages, [RE] and[ER], respectively. For more information on the format of GREIS messages, refer to “General Format of Messages” on page 280. The [RE] and [ER] messages themselves are described in “Interactive Messages” on page 320.

2.2 Language Syntax

GREIS defines *lines* of ASCII characters of arbitrary length[1], delimited by either carriage-return (<CR>, ASCII decimal code 13), or line-feed (<LF>, ASCII decimal code 10) characters, to be the top-level syntax elements of the language. Empty lines are allowed and ignored in GREIS. As a consequence, a line could be delimited by any combination of <CR> and/or <LF> characters. It allows GREIS to seamlessly support Windows™, Mac™, and UNIX™ line ending conventions.

Receiver input language is *case-sensitive*. It means that, for example, strings `GREIS`, `greis`, and `gReIs`, being different strings, are indeed considered as such by the receiver.

The number sign (`#`, ASCII code 35) is the comment introduction character. Receiver ignores everything starting from this character up to the end of the line.

After comment (if any) is stripped from the line, receiver removes leading and trailing spaces, and then breaks the line into *statements*. Statements are delimited with semicolon (`;`, ASCII code 59), or with two ampersands (`&&`, ASCII codes 38), or with two vertical bars (`||`, ASCII codes 124). Statements in a line are then executed in order, from left to right. If statement that ends in `&&` delimiter produces an error, the rest of statements in the line are not executed. If statement that ends in `||` delimiter executes successfully, the rest of statements in the line are not executed. Statement that ends in semicolon never stops execution of the sequence of statements. Note that the end of line is by itself statement terminator, so you don't need to put one of explicit statement delimiters at the end of the line.

The format of a statement is as follows:

```
[%ID%][COMMAND][@CS]
```

where square brackets denote optional fields, and any number of whitespaces is allowed before and after every field. Such whitespaces are ignored, except for the purpose of checksum calculation, see below. The fields are:

`%ID%` – statement identifier, where `ID` denotes arbitrary string, possibly empty. The identifier, if present, is copied unchanged by the receiver into the response message for the statement. Any statement with an identifier will always generate a response from the receiver. A statement that contains only an identifier is also allowed; in such a case, the receiver will just generate a response message.

`COMMAND` – a (possibly empty) *list* where the first element is called *command name*. It denotes the action to be performed. The rest of elements (if any) are command

1. Current GREIS implementation in the receivers supports lines of up to 256 characters in length.

arguments. Braces that surround command list could be omitted. Refer to “Lists” on page 15 for the syntax of lists.

`@CS` – checksum, where `CS` is 8-bit checksum formatted as 2-byte hexadecimal number. Before executing a statement with checksum, the receiver will compare the input checksum `CS` against that computed by the firmware and will refuse to execute the statement should these checksums mismatch. Checksum is computed starting with the statement's first non-blank character until and including the `@` character. See “Computing Checksums” on page 365 for details.

Statement identifier, `%ID%`, serves the following purposes:

1. Forces receiver response to the command.
2. Allows to send multiple commands with different identifiers to the receiver without waiting for response for every command, then receive the responses and tell which response corresponds to which command.
3. Helps to establish synchronization with the receiver by allowing to check that particular receiver response corresponds to particular command, and not to some other command issued before or after.

A list called *options* could be appended to any element of the `COMMAND` after the colon (`:`, ASCII code 58). If options list comprises single element, the surrounding braces could be omitted. Options list appended to a list propagates to every element of the list, though the options explicitly appended to an element of the list take precedence over propagated options. For example,

```
{e1,{e2:{o1,,o3},e3}}:{o4,o5}
```

is equivalent to:

```
{e1:{o4,o5},{e2:{o1,o5,o3},e3:{o4,o5}}}
```

Note also how missed `o2` option allows `o5` option to propagate to the list of options for `e2` element.

The number and the meaning of arguments and options in the command depends on particular command action and is defined in the description of every receiver command. In addition, if command description specifies some options, but some or all of them are missed in the statement, the default values for the missed options are substituted. The default values for options are also defined in the description of every receiver command.

For reference, below is the table comprising all the character sequences that have special meaning in the receiver input language:

Table 2-1. Input Language Special Characters

Characters	Decimal ASCII code	Meaning
<LF>	10	line separator
<CR>	13	line separator
#	35	beginning of comment mark
;	59	statements separator
&&	38	statements *and* separator
||	124	statements *or* separator
%	37	statement identifier mark
@	64	checksum mark
{	123	beginning of list mark
}	125	end of list mark
,	44	list elements separator
:	58	options mark
"	34	quotation mark
\	92	escape

2.3 Commands

In this section we describe all the commands defined in GREIS. Syntax and semantics specifications of every command are accompanied by explanatory examples. For detailed description of objects used as arguments in the examples, please refer to Chapter 3 on page 55.

2.3.1 set

Name

`set` – set value of an object.

Synopsis

Format: `set,object,value`
Options: `none`

Arguments

`object` – the target object identifier. If `object` does not begin with "`/`", then "`/par/`" prefix is automatically inserted before the `object` prior to executing the command.

`value` – the value to be assigned to the target object. The range of allowed values as well as semantics of the assignment depends on the type of the object and is specified later in this manual for every supported object.

Options

None.

Description

This command assigns `value` to the `object`. No response is generated unless there is an error or response is forced by the statement identifier.

Examples

Example: Set baud rate of serial port C to 115200. Either of:

⇒ `set,/par/dev/ser/c,115200`

⇒ `set,dev/ser/c,115200`

Example: Set baud rate of serial port A to 9600 and force reply:

⇒ `%%set,dev/ser/a/rate,9600`
⇐ `RE002%%`

●

2.3.2 print

Name

`print` – print value of an object.

Synopsis

Format: `print,object`
Options: `{names}`

Arguments

`object` – the object identifier of the object to be output. If `object` does not begin with "/", then "`/par/`" prefix is automatically inserted before the `object` prior to executing the command.

Options

Table 2-2. `print` options summary

Name	Type	Values	Default
names	boolean	`on,off`	`off`

`names` – if `off`, output only object values. When `on`, output object names in addition to object values in the format `NAME=VALUE`.

Description

This command prints value of the `object`, optionally prefixing the value with the name of corresponding object. The response is always generated, and more than one [RE] message could be generated in response to a single `print` command.

The value of an object of type *list* is printed as a list of values for every object in the list. This is applied recursively until leaf objects are reached, so printing an object of non-leaf type effectively outputs entire sub-tree starting from the specified `object`. In case of printing of lists, multiple [RE] messages could be generated. However, splitting of the output may occur only immediately after list separator characters.

Examples

Example: Print current period of the internal receiver time grid. Either of:

```
⇒ print,/par/raw/curmsint
⇐ RE004 100
```

```
⇒ print,raw/curmsint
⇐ RE004 100
```

Example: Print current period of the internal receiver time grid along with the object name. Either of:

```
⇒ print,/par/raw/curmsint:on
⇐ RE015/par/raw/curmsint=100
```

```
⇒ print,raw/curmsint:on
⇐ RE015/par/raw/curmsint=100
```

Example: Print receiver version information:

```
⇒ print,rcv/ver
⇐ RE028{"2.5 Sep,13,2006 p2",0,71,MGGDT_5,none,
⇐ RE00D {none,none}}
```

Example: Print receiver version information along with corresponding names:

```
⇒ print,rcv/ver:on
⇐ RE043/par/rcv/ver={main="2.5 Sep,13,2006 p2",boot=0,hw=71,board=MGGDT_5,
⇐ RE00C modem=none,
⇐ RE017 pow={fw=none,hw=none}}
```

Example: Print all the messages enabled for output to serial port B along with their scheduling parameters:

```
⇒ print,out/dev/ser/b:on
⇐ RE02D/par/out/dev/ser/b={jps/RT={1.00,0.00,0,0x0},
⇐ RE01A jps/SI={1.00,0.00,0,0x0},
⇐ RE01A jps/rc={1.00,0.00,0,0x0},
⇐ RE01A jps/ET={1.00,0.00,0,0x0},
⇐ RE01D nmea/GGA={10.00,5.00,0,0x0}}
```

2.3.3 list

Name

`list` – list contents of an object.

Synopsis

Format: `list[,object]`
Options: `none`

Arguments

`object` – the object identifier of the object to be output. If `object` is omitted, `/log` is assumed. If `object` does not begin with "`/`", then "`/log/`" prefix is automatically inserted before the `object` prior to executing the command.

Options

None.

Description

This command outputs names of every member of the `object`. The response is always generated, and more than one [RE] message could be generated in response to a single `list` command.

If the `object` specified is not of type *list*, empty [RE] message is generated. If the object specified is a *list,* the list of names of every object in the list is printed. This is applied recursively until leaf objects are reached, so listing an object of non-leaf type effectively outputs entire sub-tree starting from the specified `object`. In case of printing of lists, multiple [RE] messages could be generated. However, splitting of the output may occur only immediately after list separator characters.

Examples

Example: Empty reply for listing of a non-list object:

```
⇒ list,/par/rcv/ver/main
⇐ RE000
```

Example: Error reply for listing of non-existing object:

```
⇒ list,/does_not_exist
⇐ ER018{2,,wrong 1st parameter}
```

Example: Obtain a list of existing log-files. Either of

```
⇒ list,/log
⇒ list
```

will produce the same output, e.g.:

```
⇐ RE013{log1127a,log1127b}
```

Example: List all standard GREIS messages supported by the receiver:

```
⇒ list,/msg/jps
⇐ RE03D{JP,MF,PM,EV,XA,XB,ZA,ZB,YA,YB,RT,RD,ST,LT,BP,TO,DO,OO,UO,GT,
⇐ RE040 NT,GO,NO,TT,PT,SI,NN,EL,AZ,SS,FC,RC,rc,PC,pc,CP,cp,DC,CC,cc,EC,
⇐ RE040 CE,TC,R1,P1,1R,1P,r1,p1,1r,1p,D1,C1,c1,E1,1E,F1,R2,P2,2R,2P,r2,
⇐ RE040 p2,2r,2p,D2,C2,c2,E2,2E,F2,ID,PV,PO,PG,VE,VG,DP,SG,BI,SE,SM,PS,
⇐ RE040 GE,NE,GA,NA,WE,WA,WO,GS,NS,rE,rM,rV,rT,TM,MP,TR,MS,DL,TX,SP,SV,
⇐ RE031 RP,RK,BL,AP,AB,re,ha,GD,LD,RM,RS,IO,NP,LH,EE,ET}
```

Example: List all the messages in the default set of messages:

```
⇒ list,/msg/def
⇐ RE040{jps/JP,jps/MF,jps/PM,jps/EV,jps/XA,jps/XB,jps/RT,jps/RD,jps/SI,
⇐ RE040 jps/NN,jps/EL,jps/FC,jps/RC,jps/DC,jps/EC,jps/TC,jps/CP,jps/1R,
⇐ RE040 jps/1P,jps/2R,jps/2P,jps/E1,jps/D2,jps/E2,jps/SS,jps/SE,jps/PV,
⇐ RE040 jps/ST,jps/DP,jps/TO,jps/DO,jps/UO,jps/IO,jps/GE,jps/NE,jps/GA,
⇐ RE01D jps/NA,jps/WE,jps/WA,jps/WO}
```

●

2.3.4 em & out

Name

`em, out` – enable periodic output of messages.

Synopsis

Format: `em,[target],messages`
Format: `out,[target],messages`
Options: `{period, phase, count, flags}`

Arguments

`target` – any output stream. If no target is specified, the current terminal, `/cur/term`, is assumed.

`messages` – the list (either with or without surrounding braces) of message names and/or message set names to be enabled. If some of the specified names do not begin with "/", then "`/msg/`" prefix is automatically inserted before such names prior to executing the command.

Options

Table 2-3. em and out options summary

Name	Type	Values	Default
period	float	[0…86400)	-
phase	float	[0…86400)	-
count	integer	[-256…32767]	0 for em 1 for out
flags	integer	[0…0xFFFF]	-

`period, phase, count, flags` – message scheduling parameters.

Description

These commands enable periodic output of the specified `messages` into the `target`, enforcing the message scheduling parameters to be those specified by `options`. No response is generated unless there is an error, or response is forced by the statement identifier.

The `em` and `out` commands are the same except the default value of the `count` option is set to 0 for `em`, and 1 for `out`. The `out` command is just a more convenient way to request

one-time output of message(s). We will speak only about `em` in this description though everything applies to the `out` as well.

The description below expects the reader is familiar with the material in the section "Periodic Output" on page 18.

For every output stream, there is corresponding *output list* of messages[2,3] that are currently enabled to be output to the given stream. When a `message` passed as argument to `em` command is not currently in the output list, the `em` command appends specified message to the *end* of the list. When a `message` passed to `em` command is already in the output list, the `em` command just changes this message's scheduling parameters and doesn't modify message's position inside the list.

Note: As the `em` command merges the specified messages to the output list, it's often a good idea to use `dm` command to clear the output list for the given stream before issuing `em` commands.

The `em` command processes the `messages` list one message at a time, from left to right, and from the first message of message set to the last message of message set. Should it encounter a name that doesn't correspond to any supported receiver message or message set, it remembers there was an error during execution, but doesn't stop processing of the `messages` list. This way all messages from the `messages` list that could be enabled will be enabled, and only single error will be reported when one or more of the specified messages can't be enabled.

When the `em` command processes a message at hand, the final operating message scheduling parameters in the corresponding output list of messages are calculated taking into account multiple sources of information about scheduling parameters, specifically:

1. Values explicitly specified in the `options` of the `em` command.
2. The default values of options of `em` command.
3. Scheduling parameters specified for the given message as part of the corresponding message set. These are taken into account only when enabling a message by specifying message set, not an individual message.
4. Current scheduling parameters of the message in the corresponding output list (if any).
5. Default scheduling parameters specified for the given message as part of the corresponding message group.

The above sources of parameters are listed in the order of their precedence, the first one having the highest precedence, and are applied individually to each of the four scheduling parameters. Therefore, values from (1) override values from (2), the resulting value

2. For a stream `NAME`, corresponding output list is called `/par/out/NAME`
3. Current firmware has arbitrary limit for maximum number of messages in an output list set to 49.

overrides value from (3), etc. However, if some of the `F_FIX_PERIOD`, `F_FIX_PHASE`, `F_FIX_COUNT`, or `F_FIX_FLAGS` bits are set in the `flags` field of the next source, corresponding fields of this next source will not be overridden.

Examples

Example: Enable one time output of NMEA GGA message to the current terminal:

⇒ `em,,nmea/GGA:{,,1}`

The same as above, but using `out` instead of `em`:

⇒ `out,,nmea/GGA`

Example: Enable the output of the default set of messages to the current log-file A using the default output parameters. Either of:

⇒ `em,/cur/file/a,/msg/def`
⇒ `em,/cur/file/a,def`

Example: Enable output of the default set of messages to the current log-file A every 10 seconds For the other output parameters, their default values will be used:

⇒ `em,/cur/file/a,def:10`

Example: Enable output of the default set of messages to the current terminal using default output parameters. Either of:

⇒ `em,/cur/term,/msg/def`
⇒ `em,,/msg/def`
⇒ `em,,def`

Example: Enable output of GREIS messages [~~](RT) and [RD] to the current terminal. Either of:

⇒ `em,,/msg/jps/RT,/msg/jps/RD`
⇒ `em,,jps/{RT,RD}`

Example: Enable output of NMEA messages GGA and ZDA to the current terminal every 20 seconds:

⇒ `em,,nmea/{GGA,ZDA}:20`

Example: Enable output of messages [SI], [EL] and [AZ] to serial port A. Set scheduling parameters for [SI] so that interval between any two subsequent [SI] messages will be equal to 10 seconds, if they coincide, and 1 second otherwise; output only the first fifty [SI] messages. In addition, the receiver, set output interval to 2 seconds for [EL] and [AZ] messages:

⇒ `em,/dev/ser/a,jps/{SI:{1,10,50,0x2},EL,AZ}:2`

Example: Enable output of RTCM 2.x message types 1 and 31 to serial port B with output interval 3 seconds, and RTCM 2.x message types 18, 19, 3, 22 to port C with output interval 1 second for types 18 and 19; and 10 seconds for types 3 and 22:

⇒ `em,/dev/ser/b,rtcm/{1,31}:3; em,/dev/ser/c,rtcm/{18:1,19:1,22,3}:10`

●

2.3.5 dm

Name

`dm` – disable periodic output of messages.

Synopsis

Format: `dm[,[target][,messages]]`
Options: `none`

Arguments

`target` – any output stream. If no target is specified, the current terminal, `/cur/term`, is assumed. If some of the specified names do not begin with "/", then "`/msg/`" prefix is automatically inserted before such names prior to executing the command.

`messages` – the list of messages to be disabled, either with or without surrounding braces. If no messages are specified, all periodic output to the target is disabled.

Options

None.

Description

This command disables periodic output of the specified `messages` into the object `target`. No response is generated unless there is an error, or response is forced by the statement identifier.

If no messages are specified, all the periodic output to the `target` is disabled.

If the `target` is a current log-file and no messages are specified, all the output to the file is disabled, the file is closed, and corresponding current log-file is set to none.

If a message is specified in the `messages` list that is not currently enabled to be output to the given `target`, no corresponding error is generated by the `dm` command. Though this condition doesn't disable other possible errors from being reported.

Examples

Example: Disable all of the messages being output into the current log-file A and close the file:

⇒ `dm,/cur/file/a`

Example: Disable all the periodic output into the current terminal. Either of:

⇒ `dm,/cur/term`

⇒ `dm`

Example: Disable output of GREIS message [~~](RT) into the serial port B:

⇒ `dm,/dev/ser/b,/msg/jps/RT`

Example: Disable output of the GREIS message [~~] into the current log-file B:

⇒ `dm,/cur/file/b,/msg/jps/RT`

Example: Disable output of the NMEA messages GGA and ZDA into the current terminal. Either of:

⇒ `dm,/cur/term,/msg/nmea/GGA,/msg/nmea/ZDA`

⇒ `dm,,/msg/nmea/GGA,/msg/nmea/ZDA`

⇒ `dm,,nmea/GGA,nmea/ZDA`

⇒ `dm,,nmea/{GGA,ZDA}`

●

2.3.6 init

Name

`init` - initialize objects.

Synopsis

Format: `init,object[/]`
Options: `none`

Arguments

`object` - the object to be initialized.

`/` - if present and the `object` is of type *list*, initialize all the contained objects instead of the `object` itself.

Options

None.

Description

This command initializes specified objects. No response is generated unless there is an error, or response is forced by the statement identifier.

The exact semantics of initialization depends on the object being initialized, but in general could be considered as turning an object to its "default" or "clean" state. For example, for parameters it means setting their values to corresponding defaults, for the file-storage device it means re-formatting the underlying medium, etc.

Note: Initializing some of objects will result in receiver reboot. This is currently the case for initialization of receiver non-volatile memory (`/dev/nvm/a`).

Note: Though it may change in the future, current implementation of this generic command in the receivers is rather limited. In fact only initialization of objects that are found in the examples below is currently supported.

Examples

Example: Clear NVRAM and reboot receiver. All the data stored in the NVRAM (almanacs, ephemerides, etc.) will be lost, all the parameters will be set to their default values after reboot:

⇒ `init,/dev/nvm/a`

Example: Set all the receiver parameters to their default values:

⇒ `init,/par/`

Example: Initialize the file system (i.e., reformat the underlying medium). All files stored in the receiver will be lost:

⇒ `init,/dev/blk/a`

Example: Initialize all the message sets to their default values:

⇒ `init,/msg/`

2.3.7 create

Name

`create` - create a new object.

Synopsis

Format: `create[,object]`
Options: `{log}`

Arguments

`object` - object identifier of the object to be created. If `object` does not begin with "/", then "/log/" prefix is automatically inserted before the `object` prior to executing the command. If omitted, then creation of a file is assumed and an unique file name is automatically generated.

Options

Table 2-4. `create` options summary

Name	Type	Values	Default
log	string	a,b,…	a

`log` - the log-file the created file is to be assigned to. The log-file selected is `/cur/log/X`, where `X` is the value of the option[4].

Description

This command creates a new object. No response is generated unless there is an error, or response is forced by the statement identifier.

Both the location in the tree and the type of the created object are defined by the `object` argument.

Two kinds of objects could be created:

1. Files. A new file is created whenever the object identifier specifies an object in a `/log` sub-tree, or when the `object` argument is omitted.
2. Message specifiers. A new message specifier is created whenever the object identifier specifies an object in a message set (e.g., `/msg/def`).

4. Current firmware supports either one or two simultaneous log-files depending on particular receiver.

Creating Files

When creating files, the `object` argument is either omitted or has a format `/log/NAME`, where `NAME` is the name of the file to be created, and `/log/` is optional. In the former case receiver will automatically select an unique name for the file. In the latter case the `NAME` specified should be a string of up to 31 characters and should contain neither spaces nor the following characters: “`,{}()@&"/\`”.

If the file `/log/NAME` already exists, the `create` command will fail and produce an error message. As a consequence, there is no way to clobber some of existing files with the `create` command.

After a new file is successfully created, it’s assigned to one of the current log-files depending on the value of the `log_file` option. If corresponding log-file already points to another file when `create` is executed, the old log-file will be closed and the output will continue into the new file without any interruption.

An automatically generated file name has the following format:

P…PMMDDS…S

where P…P is prefix specified by the value of the receiver parameter `/par/cmd/create/pre/X` (`X=log`), MM is the month number, DD is day of month number, and S…S is a suffix to make unique names for the files created on the same day. The suffix begins with empty value, then the letters `a` to `z` are assigned for every next file created on the same day, then yet another letter is appended to the suffix, etc. Overall, the suffixes go in sequence like this: `<empty>,a,b,…,z,za,zb,…zz,zza,zzb,…`. For example, an automatically generated file name may look like this: `log0917zzy`.

Creating Message Specifiers

When adding messages to a message set, the `object` argument has a format `/msg/SET/GROUP/MSG`, where `SET` is the name of the message set where new message should be created, `GROUP` is the name of the group the message belongs to, and `MSG` is the name of the message itself (e.g., `/msg/def/nmea/GGA`, or `/msg/jps/rtk/min/jps/ET`).

The message scheduling parameters will be copied from those defined for given message in the message group. Use `set` command to customize the scheduling parameters if required.

Examples

Creating Files

Example: Create a new file with an automatically generated name and assign it to the current log-file A (`/cur/file/a`). Either of:

⇒ `create`

⇒ `create,:a`

Example: Create a new log-file with the name “my_file”. Either of:

⇒ `create,/log/my_file:a`

⇒ `create,my_file`

Example: Create files “file1” and “file2”, and assign them to `/cur/file/a` and `/cur/file/b`:

⇒ `create,file1:a; create,file2:b`

●

Creating Message Specifiers

Example: Add `/msg/jps/ET` messages to the default set of messages:

⇒ `create,/msg/def/jps/ET`

Example: Add NMEA GGA message to the default set of messages and force its `period` and `phase` to be always 10 and 5, respectively, no matter what values for them will be specified in a `em` or `out` command:

⇒ `create,/msg/def/nmea/GGA`

⇒ `set,/msg/def/nmea/GGA,{10,5,,0x30}`

●

2.3.8 remove

Name

`remove` – remove an object.

Synopsis

Format: `remove,object[/]`
Options: `none`

Arguments

`object` – object identifier of the object to be removed. If `object` does not begin with "`/`", then "`/log/`" prefix is automatically inserted before the `object` prior to executing the command.

`/` – if present and the `object` is of type *list*, remove all the object contents instead of the `object` itself.

Options

None.

Description

This command removes (deletes) an existing object. No response is generated unless there is an error, or response is forced by the statement identifier.

If there is no object specified by `object`, or if the object can't be removed, an error is generated.

Two kinds of objects could be removed:

1. Files. If the file is one of current log-files, the command will fail and error message will be generated.
2. Message specifiers from message sets.

Examples

Example: Remove the log-file with the name "NAME". Either of:

⇒ `remove,/log/NAME`

⇒ `remove,NAME`

Example: Remove all log-files:

⇒ `remove,/log/`

Example: Remove GREIS standard [GA] message from the default set of messages:

⇒ `remove,/msg/def/jps/GA`

Example: Remove all the messages from the default set of messages:

⇒ `remove,/msg/def/`

Example: Remove all the messages from the minimal set of standard GREIS messages suitable for RTK:

⇒ `remove,/msg/rtk/jps/min/`

2.3.9 event

Name

`event` - generate free-form event.

Synopsis

Format: `event,string`
Options: `none`

Arguments

`string` - an arbitrary[5] string comprising up to 63 characters.

Options

None.

Description

This command generates a free-form event. No response is generated unless there is an error, or response is forced by the statement identifier.

The given `string` along with the time of receiving the `event` command is stored in the receiver in the special event buffer[6]. The contents of this buffer is output to all the output streams where the standard GREIS message [==](EV) (described on page 321) is enabled.

The free-form event mechanism is intended for the control programs to forward arbitrary text information to post-processing applications without interpreting this information in the receiver. The receiver firmware's core never generates free-form events on its own, nor does it somehow interpret the information sent through the `event` commands.

Note: All of the strings starting with the underscore character (ASCII 0x5F) are reserved for JAVAD GNSS applications. Care should be taken that such strings are not used with the `event` commands unless you can't accomplish your task otherwise or intend to cooperate with some JAVAD GNSS software. In the latter case please refer to detailed description of free-form events reserved for JAVAD GNSS applications in the *"Frame Format for Free-Form Events"* guide, available from *http://www.javad.com*.

Example: Generate a free-form event containing the string "Info1":

⇒ `event,Info1`

5. Recall that if a string contains any of the characters reserved for the receiver input language, you should enclose this string in double quotes.
6. The current firmware provides a buffer large enough to store up to sixteen 64 byte free-form events.

Example: Generate a free-form event containing reserved characters:

```
⇒ event,"EVENT{DATA,SENT}"
```

Example: Generate free-form event reserved for JAVAD GNSS application software (this event notifies postprocessing application about change of dynamics):

```
⇒ event,"_DYN=STATIC"
```

Example: Generate a free-form with empty string:

```
⇒ event,""
```

Example: Generate a few free-form events and get back the [==](EV) messages (in the contents of [==] messages non-printable bytes are replaced with dots in the example):

```
⇒ em,,jps/EV
⇒ %accepted% event,"some string"
⇐ RE00A%accepted%
⇐ ==011.....some_string.
⇒ %1% event,1; %2% event,2
⇐ RE003%1%
⇐ RE003%2%
⇐ ==007.....1.
⇐ ==007.....2.
⇒ dm,,jps/EV
```

●

2.3.10 get

Name

get – start retrieving of file contents using DTP[7].

Synopsis

Format: get,object[,offset]
Options: {timeout,block_size,period,phase,attempts}

Arguments

object – object identifier of the file to be retrieved. If object does not begin with "/", then "/log/" prefix is automatically inserted before the object prior to executing the command. If the object does not exist or can't be retrieved, an error message is generated.

offset – offset in bytes from the beginning of the file at which to start retrieving. If omitted, 0 is assumed.

Options

Table 2-5. get options summary

Name	Type	Values	Default
timeout	integer	[0…86400], seconds	10
block_size	integer	[1…16384[1]]	512
period	float	[0…86400), seconds	0
phase	float	[0…86400), seconds	0
attempts	integer	[1…100]	10

1. 2048 for receivers that don't support TCP or USB.

timeout – the timeout for DTP.

block_size – the size of a DTP data block.

period – the output period for filtering (see below).

phase – the output phase for filtering (see below).

attempts – maximum number of attempts DTP transmitter will take to send single block. When set to 1, special *streaming* mode is activated (see below).

7. See "Data Transfer Protocol" on page 366.

Description

This command starts retrieving of a file into the host computer using the Data Transfer Protocol (DTP). No response is generated unless there is an error, or response is forced by the statement identifier.

After the `get` command succeeds, the *DTP transmitter* is started on the receiver and waits for *DTP receiver* to be run on the host. Therefore, to actually retrieve any data, one needs *DTP receiver* implementation on the host.

Note: The essential difference between the `get` and `print` commands is that the former is based on the data transfer protocol that provides error detection and correction, whereas the latter just blindly outputs data into an output stream.

The optional `offset` argument allows host to implement support for resuming of interrupted data transfer. Note that seeking to a large offset may require rather long time to perform in the receiver. To correctly implement resumption in the host software, force receiver response to the `get` command using *statement identifier* and wait for the reply from the receiver before running DTP on the host. This method takes advantage of the fact that receiver replies to the `get` command *after* the seek is performed.

When the `attempts` option is set to `1`, the *DTP transmitter* will be put into so-called *streaming* mode. In this mode, after receiving the first `NACK` from the *DTP receiver*, the *DTP transmitter* will stream data blocks without waiting for `ACK`s from the *DTP receiver*, and the *transmitter* will immediately abort data transfer should `NACK` be received. This approach allows significantly faster data transfer over reliable connections having high latencies (such as TCP) or relatively high direction switch overhead (such as USB). Correctly implemented receiving part of the protocol does not require any special care to support this method.

When the `period` option is non-zero special *filtering* mode is activated. For example, it allows to download 1Hz data from a file that was written using 10Hz update rate. Specifically, the receiver will send the data only for the epochs where receiver time modulo one day (T_r) satisfies the following equation:

$$T_r \{\mathrm{mod}\ \texttt{period}\} = \texttt{phase}$$

To achieve this, receiver parses the contents of the file and filters-out some of the messages. Note that implementation of resumption of interrupted download is very hard if not impossible in this case due to the fact that the host has no idea at what offset of the receiver file the download has been interrupted.

Examples

Example: Start retrieving the contents of the file “NAME” using DTP. Either of:

```
⇒ get,/log/NAME
⇒ get,NAME
```

Example: Start retrieving the contents of the file “NAME” starting from byte number 3870034 (counting bytes from zero). Expect rather long time to pass between the command and the reply:

```
⇒ %%get,/log/NAME,3870034
⇐ RE002%%
```

Example: Start retrieving the contents of the file “my_logfile” starting from byte 3000 using timeout 50 seconds and block size of 8192 bytes:

```
⇒ get,my_logfile:{50,8192},3000
```

Example: Start retrieving the contents of the file “NAME” filtering out epochs so that the resulting retrieved file would be 0.1Hz data:

```
⇒ get,NAME:{,,10}
```

Example: Start retrieving the contents of the file “NAME” using “streaming” mode (`attempts` option set to 1):

```
⇒ get,NAME:{,,,,1}
```

●

2.3.11 put

Name

`put` – start file uploading using DTP[8].

Synopsis

Format: `put,object[,offset]`
Options: `{timeout, block_size}`

Arguments

`object` – object identifier of the file to write data to. If `object` does not begin with "`/`", then "`/log/`" prefix is automatically inserted before the `object` prior to executing the command.

`offset` – offset in bytes from the beginning of the file at which to start writing. If omitted, 0 is assumed.

Options

Table 2-6. `put` options summary

Name	Type	Values	Default
`timeout`	integer	`[0…86400]`, seconds	`10`
`block_size`	integer	`[1…16384`[1]`]`	`512`

1. 2048 for receivers that don't support TCP or USB.

`timeout` – the timeout for DTP.

`block_size` – the size of a DTP data block.

Description

This command starts uploading of data from host computer into a file in the receiver using the Data Transfer Protocol (DTP). No response is generated unless there is an error, or response is forced by the statement identifier.

After the `put` command succeeds, the *DTP receiver* is started on the receiver and waits for *DTP transmitter* to be run on the host. Therefore, to actually upload any data, one needs *DTP transmitter* implementation on the host.

8. See "Data Transfer Protocol" on page 366.

The optional `offset` argument allows host to implement support for resuming of interrupted data transfer. A non-zero `offset` value allows host to request appending data to the end of an existing file of matching size.

If `offset` is 0 and the file `object` doesn't exist, receiver will try to create and open for writing a new file with the name defined by `object`. In this case the command will fail if there already exist a file with given name.

If offset is greater than 0, and there is a file `object`, and the file size is equal to the value of `offset`, then the `put` command will open the file `object` for append. In this case the command will fail if there is no existing file with given name or if the size of the existing file doesn't match those specified by `offset`.

Examples

Example: Start data uploading to a fresh file "NAME" using DTP. Either of:

⇒ `put,/log/NAME`

⇒ `put,NAME`

Example: Start uploading data and append them to existing file "NAME". Use default DTP timeout and DTP block size 4096 bytes. Get the size of the file before starting the upload (note that the file size is required on host anyway so that it can skip this number of bytes from its source data file):

⇒ `print,/log/NAME&size`
⇐ `RE008 3870034`

⇒ `put,/log/NAME:{,4096},3870034`

Example: Start data uploading to a fresh file "my_logfile" using timeout 50 seconds and block size of 8192 bytes:

⇒ `put,my_logfile:{50,8192}`

●

2.3.12 fld

Name

`fld` – firmware loading.

Synopsis

Format: `fld,id,object`
Options: `{timeout, block_size}`

Arguments

`id` – string containing the receiver electronic ID[9]. If specified ID does not correspond to the actual electronic ID of the receiver, the command will fail and produce error message.

`object` – object identifier of the source of the firmware to be loaded. Either the name of receiver file, or the name of an input port[10].

Options

Table 2-7. `fld` options summary

Name	Type	Values	Default
timeout	integer	[0…86400], seconds	10
block_size	integer	[1…16384[1]]	512

1. 2048 for receivers that don't support TCP or USB.

`timeout` – the timeout for DTP.

`block_size` – the size of a DTP data block.

Description

This command loads firmware from specified `object` into receiver and then resets the receiver. No response is generated unless there is an error, or response is forced by the statement identifier.

Warning: *Should a power failure or interruption of firmware transfer through a port occur during the loading, the receiver will go into a non-working state where only firmware loading through RS-232 ports using "power-on capture" method is possible.*

9. The ID could be obtained using `print,/par/rcv/id` command.
10. Current firmware supports only `/dev/usb/a` as an input stream for firmware loading.

If the `object` designates an existing file[11], the receiver will first check whether the file contains valid firmware for the receiver (it takes a number of seconds to complete). If the check succeeds, the receiver will load the firmware and then perform self-reset. Note that the reply to the command (if any) will be sent after the check is performed but before firmware loading begins. The `timeout` and `block_size` options are ignored in this case.

If `object` designates an input stream, the command will send the reply (if any) and then start *DTP receiver* that will wait for *DTP transmitter* to be run on the host. Therefore, to actually upload the firmware, one needs *DTP transmitter* implementation on the host. Self reset (reboot) will be performed by the receiver after the loading successfully completes or is interrupted.

Examples

Example: Load firmware from the file "firmware.ldp" into receiver with electronic ID `123456789AB`. Expect a few seconds to pass between sending the command and receiving reply, while receiver checks the file for firmware validity:

```
⇒ %%fld,123456789AB,/log/firmware.ldp
⇐ RE002%%
```

Example: Start firmware uploading from the USB port using block size 16384 bytes and timeout 20 seconds. Obtain electronic ID before issuing the command:

```
⇒ print,rcv/id
⇐ RE00C 8PZFM10IL8G
⇒ fld,8PZFM10IL8G,/dev/usb/a:{20,16384}
```

●

11. It is expected that the file containing the firmware is uploaded to the receiver in advance, e.g., using the `put` command.

Receiver Objects

In this chapter we will describe all the receiver objects in details.

3.1 Overview

Recall that every object has an unique identifier, or *name*, that is used to address the object in GREIS commands, and that all the objects are organized into single tree-like structure that not only groups related objects together, but also allows to apply a command to a group of objects. The object tree starts at the single root list, and ends at the tree leafs. As all non-leaf objects have the same type *list* and behave similarly with respect to GREIS commands[1], we mostly describe leaf objects in this chapter.

Most of the (leaf) objects could be used both in the `print` (or `list`) and `set` commands. We call such objects *read-write* objects. Those objects that can't be used in the `set` command, are called *read-only* objects, whereas objects that can't be used in the `print` command, are called *write-only* objects. Description of each object contains the field *access* that specifies if the object is read-write or read-only. If an object could be used as an argument in some other GREIS commands, this ability will be explicitly mentioned in the description of the object.

Each object has a *type* associated with it. Object type defines the formats that are accepted by the `set` command for this object, and the format that the `print` command will use when it reports the state of the object. Note that the `set` command may accept multiple formats for given type, whereas the `print` command will always use one fixed format from those supported by the `set` command. For example, the `set` command for an *integer* type will accept values in decimal, hexadecimal, or octal format, while the `print` command will always use one of these formats for given object. The format that is typically used by the `print` command for a given type is called the *default format* for this type. Should the `print` command use non-default format for an object, the format is either explicitly specified in the description of this object, or matches those that is used to specify the allowed values and the default value of the object.

1. Due to limitations of the current implementation of the `set` command, it doesn't support most of non-leaf objects. Those non-leaf objects that nevertheless are supported by the `set` command are explicitly described in this chapter.

When appropriate, an object description contains a range (or a list) of allowed values, as well as the default value of the object. The allowed values and the default value are always specified in the format that the `print` command will use for this object.

3.2 Conventions

3.2.1 Object Specification

Every object specification found in the section “Objects Reference” on page 64 has the following representation:

Name:	`name`
Access:	`access`
Type:	`type`
Values:	`allowed_values`
Default:	`default_value`
Options:	`options_spec`

<DESCRIPTION>

where:

`name` – is the full name of the object (object identifier).

`access` – access type. `rw` – for read-write object, `r` – for read-only object, or `w` – for write-only object.

`type` – the type of the object and the measurement units of the object, the latter being taken into square brackets.

`allowed_values` – specification of the range of values allowed for the object. For integer or float values, the range is specified in the form `[A…B]`, where `A` and `B` are the lower and upper bounds of the range, inclusive. If a bound is excluded, then round bracket is used instead of square one, e.g., `[A…B)` means the range where `A` is included and `B` is excluded. For a list of allowed values, the values listed are delimited either by comma or by the vertical bar (`|`) character.

`default_value` – the default value of the object in the format that the `print` command will use, or the text `(empty string)` that for objects of type string denotes the string comprising zero characters.

`options_spec` – the specification of options the `set` command may take for given object in the format `{op1: type1,…,opN: typeN}`, where `opX` is the description of option, and `typeX` is corresponding option type.

`<DESCRIPTION>` – textual description of the object and the meaning of its values.

3.2.2 Input and Output Ports Notations

Receiver may support many input/output ports. To denote receiver ports in the object specifications, the notations described in this section are used.

[port] – input/output port

The `[port]` denotes any of ports suitable both for input and for output. It may take one of the following values:

`dev/ser/X, X=[a…d]` – RS232 (serial) ports

`dev/tcp/X, X=[a…e]` – raw TCP ports

`dev/usb/a` – USB port

`dev/tcpcl/a` – TCP client port

`dev/can/X, X=[a,b]` – CAN ports

`dev/prl/a` – parallel port (almost obsolete, – no newer receivers support this)

[oport] – output port

The `[oport]` denotes any of ports suitable for output only. It may take one of the following values:

`[port]` – input/output ports

`cur/file/X, X=[a,b]` – current log-files

`dev/udp/a` – UDP port

cur/term – current terminal

You can use the string `cur/term` to denote the port the command is issued by wherever `[port]` or `[oport]` is allowed. The `cur/term` will be substituted by the actual port name before the command is executed. Therefore, for example, if you set some parameter to `/cur/term` when sending command through `/dev/ser/a`, the value of the parameter, once the command is executed, will become `/dev/ser/a`.

3.3 Primary Object Types

In this section, for object types that are frequently used, we describe formats that are accepted by the `set` command and could be used by the `print` commands. Formats for

the object types that are used for single object are described along with corresponding objects.

3.3.1 list

The `list` format is a comma-separated sequence of fields surrounded by braces (`{` and `}`). When an object of this type is assigned a value using `set` command, some of the fields could be omitted, in which case corresponding fields will retain their previous values.

3.3.2 array

The type `array` is a kind of type `list` where all the fields have the same type and have names assigned after the decimal representations of their indexes.

The notation

```
array [N…M] of <type>
```

is used in descriptions of array objects, where:

`N` – is the index of the first element of the array

`M` – is the index of the last element of the array

`<type>` – is the name of the type of elements of the array

The format for `array` type is the same as for `list` type.

For arrays of `boolean`, in addition to the format of the `list` type, the value for the `set` command could be given as an integer number, where the bits of the number correspond to elements of the array. Least significant bit (bit #0) corresponds to the first element of the array, and bit #`K`, where `K=M-N`, corresponds to the last element of the array. If bit is set to 1, corresponding element will have *true* value; if bit is set to 0, corresponding element will have `false` value.

3.3.3 integer

Integer values could be specified in one of the following formats:

`decimal` – optional plus or minus sign, then one or more digits in the range `[0…9]`, where the first digit is not `0`. For example, `493`.

`octal` – optional plus or minus sign, then the digit `0` followed by one or more digits in the range `[0…7]`. For example, `0371`.

`hexadecimal` – optional plus or minus sign, then the string `0x` followed by one or more characters in the range `[0…9,a…f,A…F]`. For example, `0x03f0`, or `-0xCAF`.

Decimal representation is the default one for `print` command.

3.3.4 float

Float values could be specified in the following format:

- An optional plus or minus sign (`+` or `-`).
- A nonempty sequence of digits optionally containing a decimal-point character (`.`).
- An optional exponent part, consisting of a character `e` or `E`, an optional sign, and a sequence of digits.

For example, `+3.24e-10`, or `-0.001`.

3.3.5 enumerated

An object of type `enumerated` may posses one of the values taken from predefined set of value. The set of possible values is defined for each object of this type individually.

3.3.6 boolean

An object of type boolean may posses the following values:

`y, yes, on` – stand for *true*

`n, no, off` – stand for *false*

The `print` command will use variant specified in the parameter description.

3.3.7 string

The type `string` denotes possibly empty sequence of characters. When upper- and lower-number of characters in the string is defined, the notation `string [N…M]` is used, where `N` is the minimum, and `M` is the maximum allowed number of characters in the string.

3.3.8 sched_params

The type `sched_params` denotes message scheduling parameters. The supported format is as follows:

```
{period,phase,count,flags}
```

where:

`period` – field of type `float` denoting message output period in seconds within the range `[0…86400)`.

`phase` – field of type `float` denoting message output phase or forced output period in seconds within the range `[0…86400)`.

`count` – message output count of type `integer` in the range `[-256…32767)`.

`flags` – message scheduling flags of type `integer` formatted as hexadecimal.

For detailed description of the message scheduling parameters, refer to “Periodic Output” on page 18.

3.3.9 timespec

The `timespec` type is used to define time specifications. The canonical form of `timespec` is as follows:

```
DdHHhMMmSSs
```

where `D`, `HH`, `MM`, and `SS` fields are either integer numbers in corresponding range or special value “__” serving as wild-card.

`D` – number of day inside a week `[1…7]`. `1`-Sunday… `7`-Saturday.

`HH` – number of hour inside a day `[0…23]`.

`MM` – number of minute inside an hour `[0…59]`.

`SS` – number of second inside a minute `[0…59]`.

Here are two examples of valid timespec: `4d17h40m18s` `__d__h00m__s`

Receivers `print` command always outputs timespec in its canonical form. Receivers `set` command, however, accepts timespec not only in canonical form, but also in simplified forms. The rules for `set` command are as follows:

- If some field is omitted, it is assumed to be “__”.
- Single underscore is the same as double underscore.
- Any number of integer digits is accepted.

Thus, for example, empty string `timespec` is taken as `__d__h__m__s`, and `8h2s` is taken as `__d08h__m02s`.

3.3.10 ip_address

The `ip_address` type format is standard Internet IPv4 address in numbers-and-dots notation.

3.3.11 datum_id

The `datum_id` type is a string of up to 5 upper-case characters designating datum identifier. Refer to “Datums” on page 89 for details.

3.3.12 pos_xyz

The `pos_xyz` type is used to denote Cartesian coordinates. It has the following format:

```
{datum_id,x,y,z}
```

where:

`datum_id` – field of the type `datum_id` specifying the datum to which coordinates are referenced.

`x,y,z` – Cartesian coordinates in meters. Allowed range is `[-10000000…10000000]`.

3.3.13 pos_geo

The `pos_geo` type is used to denote Geodetic coordinates. It has the following format:

```
{datum_id,lat,lon,alt}
```

where:

`lat` – latitude (see below for accepted formats). Allowed range is `[-90…90]` degrees. Negative latitude corresponds to the Southern hemisphere.

`lon` – longitude (see below for accepted formats). Allowed range is `[-180…180]` degrees. Negative longitude corresponds to the Western hemisphere

`alt` – altitude in meters. Allowed range is `[-20000…20000]`.

Output Format for Angles

For latitude and longitude, the `print` command uses the following formats:

`[N|S]DDdMMmSS.SSSSSSs` – for latitude

`[E|W]DDDdMMmSS.SSSSSSs` – for longitude

where

`N, S, E, and W` – designate Northern, Southern, Eastern, and Western hemisphere, respectively,

`DD, DDD` – integer degrees,

`MM` – integer minutes,

`SS.SSSSSS` – integer and fractional seconds,

`d, m, s` – delimiters.

For example,

`N83d42m47.556000s` – means 83 degrees 42 minutes and 47.556 seconds Northern latitude.

`E083d42m47.556000s` – means 83 degrees 42 minutes and 47.556 seconds Eastern longitude.

The `set` command supports two different formats for latitude and longitude. These are the *general format* and the *almost fixed format*.

General Input Format for Angles

General format for entering latitudes and longitudes is an extended version of the format that receiver uses when it outputs these angles.

As the name of this format implies, this is a very flexible format enabling you to specify latitude and longitude in a number of different ways. You can use various angular units (specifically, degrees, minutes, seconds, and radians) and their combinations.

Angle representation may comprise one or more floating point numbers. Every float number in the angle representation except the right-most one must have a delimiter after it. Allowed delimiters are “`d`”, “`m`”, “`s`”, or “`r`”, which denote degrees, minutes, seconds, and radians, respectively. Using a delimiter after the right-most float number in the angle representation is optional. If you omit the delimiter after the right-most float number, the

receiver will first identify the preceding delimiter and then retrieve the omitted one by using the following decision rule:

Preceding Delimiter	Assumed Omitted Delimiter
none	d
d	m
m	s
s	s
r	r

An angle representation may or may not have a direction sign. Direction signs are “E” and “W” to denote Eastern and Western longitude, “N” and “S” to denote Northern and Southern latitude, respectively. A direction sign may be placed either at the very beginning or at the very end of the angle representation (see the examples below).

Each of the separate floats describing “degrees”, “minutes”, “seconds” and “radians” is multiplied by an appropriate factor and the resulting products are accumulated. Also, if there is either “W” or “S” in the angle representation, the resulting sum's sign is inverted. If there is no direction sign in the angle representation, such notation is still valid.

Example: `37.87` and `37.87d` are equivalent. Either means 37.87 degrees Eastern longitude or 37.87 degrees Northern latitude.

Example: `37.87W`, `W37.87`, `37.87dW`, and `W37.87d` are all equivalent meaning 37.87 degrees Western longitude.

Example: `27d37m20.45sE` and `27d37m20.45E` are equivalent representations meaning 27 degrees 37 minutes and 20.45 seconds Eastern longitude.

Example: `-0.85r` means either 0.85 radians Western longitude or 0.85 radians Southern latitude.

Example: `27.13d-12.6s34dW` means 27.13 degrees minus 12.6 seconds plus 34 degrees, Western longitude. It is equivalent to 61.13 degrees minus 12.6 seconds, Western longitude; or simply 61.1265 degrees Western longitude.

Almost Fixed Input Format for Angles

This format is modeled after the format used to represent latitudes and longitudes in the NMEA sentences.

This format comprises the following data fields (from left to right):

- letter “x” (in lower case),
- one or more decimal digits before the decimal point,

- optional decimal point ".",
- zero or more decimal digits after the decimal point,
- direction sign ("`E`" or "`W`" for longitude, "`N`" or "`S`" for latitude).

The following rules are used to extract integer degrees, integer minutes, and fractional minutes from this format:

- Digits after decimal point (if any) denote fractional part of minutes.
- Up to two decimal digits immediately before the decimal point (or immediately before the "`E`", "`W`", "`N`" or "`S`" signs provided there is no decimal point), denote integer number of minutes.
- The remaining decimal digits (the leftmost ones) denote integer number of degrees.

Example: `x12023.234E` means 120 degrees and 23.234 minutes Eastern longitude.

Example: `x1202E` means 12 degrees and 2 minutes Eastern longitude.

3.4 Objects Reference

This section contains the complete list of objects used to control the operation and behavior of the receiver.

3.4.1 Power Management

Reset receiver

Name:	`/par/reset`
Access:	`w`
Type:	`boolean`
Values:	`yes,no`
Default:	`no`

`yes` – setting this parameters to `yes` will reset (reboot) the receiver. From a functional point of view, the reset is equivalent to turning the power off and then back on. The value of this parameter will be automatically returned back to `no` after the reset.

`no` – setting to `no` is ignored.

Power Off

Name: `/par/power`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `on`

`on` – setting this parameter to `on` is silently ignored.

`off` – Setting this parameter to `off` turns the receiver off. There is no way to turn the receiver on after that using GREIS commands, and after receiver is turned on by other means (e.g., using a power button), this parameter returns back to `on`.

Sleep Mode

Name: `/par/sleep`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `off`
Options: `{wakeup_time: timespec}`

`on` – put receiver to sleep mode.

`off` – ignored.

When in sleep mode, receiver could be woken up by one of the methods supported by particular receiver model. For example, some or all receiver serial ports may be able to wake up receiver whenever some character is received. Pushing the power button will wake up receiver from the sleep mode as well.

If `wakeup_time` option is specified, receiver will be woken up on specified date and time, unless it's woken up earlier by other means.

Example: Put receiver into sleep mode so that it will wake up on Monday (2d) at 23h3m55s GPS time.

⇒ `set,/par/sleep,on:2d23h3m55s`

Low Power Mode

Name: `/par/lpm`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `on`

`on` – enable processor to enter low power mode when idling.

`off` – disable processor to enter low power mode when idling.

3.4.2 Receiver Information

Receiver Serial Number

Name: `/par/rcv/sn`
Access: `r`
Type: `string[0…31]`

This parameter contains serial number assigned to the receiver on the factory.

Receiver Electronic ID

Name: `/par/rcv/id`
Access: `r`
Type: `string[11]`

This parameter contains a piece of text uniquely identifying your receiver.

Receiver Model

Name: `/par/rcv/model`
Access: `r`
Type: `string`

The model of the receiver, e.g., `MAXOR`.

Receiver Vendor

Name: `/par/rcv/vendor`
Access: `r`
Type: `enumerated`
Values: `JAVAD,UNKNOWN`

Receiver Up-time

Name: `/par/rcv/uptime`
Access: `r`
Type: `timespec`

Time elapsed since last receiver reboot.

Receiver RAM Size

Name: `/par/rcv/mem`
Access: `r`
Type: `integer [kilobytes]`

Receiver Configuration Word

Name: `/par/rcv/cfgw`
Access: `r`
Type: `integer`

The receiver configuration word formatted as hexadecimal, or empty if not available.

3.4.3 Version Information

Hardware Version

Name: `/par/rcv/ver/hw`
Access: `r`
Type: `integer`

Boot-loader Version

Name: `/par/rcv/ver/boot`
Access: `r`
Type: `integer`

Firmware Version

Name: `/par/rcv/ver/main`
Access: `r`
Type: `string`

Board Version

Name: `/par/rcv/ver/board`
Access: `r`
Type: `integer`

Internal modem board version

Name: `/par/rcv/ver/modem`
Access: `r`
Type: `string`

If internal modem is not supported, the value of this parameter will be `none`.

3.4.4 Measurements Parameters

Satellites Locking Parameters

Enable Tracking of GPS Satellites by Their Numbers

Name:	`/par/lock/gps/sat`
Access:	`rw`
Type:	`array [1…32] of boolean`
Values:	`{y|n,…,y|n}`
Default:	`{y,…,y}`

Enables/disables the receiver to track GPS satellites by their PRN.

Enable Tracking of GPS Satellite Number N

Name:	`/par/lock/gps/sat/N (N=[1…32])`
Access:	`rw`
Type:	`boolean`
Values:	`y,n`
Default:	`y`

`y` – enable tracking of GPS satellite number `N`.

`n` – disable tracking of GPS satellite number `N`.

Enable Tracking of GPS Pseudolites by Their Numbers

Name:	`/par/lock/gpsext/sat`
Access:	`rw`
Type:	`array [33…37] of boolean`
Values:	`{y|n,…,y|n}`
Default:	`{n,…,n}`

Enables/disables the receiver to track GPS pseudolites by their PRN.

Enable Tracking of GPS Pseudolite Number N

Name:	`/par/lock/gpsext/sat/N (N=[33…37])`
Access:	`rw`
Type:	`boolean`
Values:	`y,n`
Default:	`n`

`y` – enable tracking of GPS pseudolite number `N`.

`n` – disable tracking of GPS pseudolite number `N`.

Enable Tracking of GALILEO Satellites by Their Numbers

Name: `/par/lock/gal/sat`
Access: `rw`
Type: `array [1…32] of boolean`
Values: `{y|n,…,y|n}`
Default: `{y,…,y}`

Enables/disables the receiver to track GALILEO satellites by their PRN.

Enable Tracking of GALILEO Satellite Number N

Name: `/par/lock/gal/sat/N (N=[1…32])`
Access: `rw`
Type: `boolean`
Values: `y,n`
Default: `y`

`y` – enable tracking of GALILEO satellite number `N`.

`n` – disable tracking of GALILEO satellite number `N`.

Types of GALILEO Satellites by Their Numbers

Name: `/par/lock/gal/giove`
Access: `rw`
Type: `array [1…32] of boolean`
Values: `{y|n,…,y|n}`
Default: `{y,…,y}`

Specifies types of GALILEO satellites by their PRN.

Type of GALILEO Satellite Number N

Name: `/par/lock/gal/giove/N (N=[1…32])`
Access: `rw`
Type: `boolean`
Values: `y,n`
Default: `y`

`y` – satellite number N is Giove satellite.

`n` – satellite number N is true GALILEO satellite.

Enable Tracking of GLONASS Satellites by FCN

Name: `/par/lock/glo/fcn`
Access: `rw`
Type: `array [-7…6] of boolean`
Values: `{y|n,…,y|n}`
Default: `{n,n,n,n,y,…,y}`

Enables/disables the receiver to track GLONASS satellites by their FCN.

Enable Tracking of GLONASS Satellite FCN #N

Name: `/par/lock/glo/fcn/N (N=[-7…6])`
Access: `rw`
Type: `boolean`
Values: `y,n`
Default: `n (N=[-7…-4]), y (N=[-3…6])`

`y` – enable tracking of GLONASS satellite with FCN `N`.

`n` – disable tracking of GLONASS satellite with FCN `N`.

Enable Tracking of SBAS Satellites by Their Numbers

Name: `/par/lock/sbas/sat`
Access: `rw`
Type: `array [120…138] of boolean`
Values: `{y|n,…,y|n}`
Default: `sensible at the moment of firmware release`

Enables/disables the receiver to track SBAS satellites by their PRN.

Enable Tracking of SBAS Satellite Number N

Name: `/par/lock/sbas/sat/N (N=[120…138])`
Access: `rw`
Type: `boolean`
Values: `y,n`
Default: `sensible at the moment of firmware release`

`y` – enable tracking of SBAS satellite number `N`.

`n` – disable tracking of SBAS satellite number `N`.

Elevation Mask for Tracking

Name: `/par/lock/elm`
Access: `rw`
Type: `integer [degrees]`
Values: `[-90…90]`
Default: `-1`

Receiver will not track satellites below this elevation mask.

P-code Tracking

Name: `/par/lock/pcode`
Access: `rw`
Type: `boolean`
Values: `always,on,off`
Default: `on`

`always` – enable P-code tracking, unconditionally.

`on` – synonym for `always`.

`off` – disable P-code tracking.

L2C Tracking

Name: `/par/lock/l2c`
Access: `rw`
Type: `boolean`
Values: `always,on,off`
Default: `on`

`always` – enable L2C tracking, unconditionally.

`on` – enable L2C tracking, provided L2C is available for given SV according to almanach.

`off` – disable L2C tracking.

L5 Tracking

Name: `/par/lock/l5`
Access: `rw`
Type: `boolean`
Values: `always,on,off`
Default: `on`

`always` – enable L5 tracking, unconditionally.

`on` – enable L5 tracking, provided L5 is available for given SV according to almanach.

`off` – disable L5 tracking.

Maximum Velocity

Name: `/par/lock/vmax`
Access: `rw`
Type: `integer [m/s]`
Values: `[1000…10000]`
Default: `1000`

This parameter specifies maximum possible velocity of the receiver antenna for the purpose of computation of the required SVs search zone.

Maximum Acceleration

Name: /par/lock/amax
Access: rw
Type: integer [m/s/s]
Values: [10…100]
Default: 20

This parameter specifies maximum possible acceleration of the receiver antenna for the purpose of computation of the required SVs search zone.

Antenna Tracking Masks

Name: /par/lock/ant
Access: rw
Type: array [1…4] of boolean
Values: {y|n,y|n,y|n,y|n}
Default: {y,y,y,y}

Each element of the array enables tracking of SVs on corresponding antenna.

Note: This parameter is only available for multi-antenna receivers.

Antenna N Tracking Mask

Name: /par/lock/ant/N (N=[1…4])
Access: rw
Type: boolean
Values: y,n
Default: y

This parameter enables tracking of SVs on antenna N.

Note: This parameter is only available for multi-antenna receivers.

Generic Measurements Parameters

Measurements Update Rate

Name: /par/raw/msint
Access: rw
Type: integer [milliseconds]
Values: [50,100,150,…,5000]
Default: 100

This parameter specifies the required period of the internal receiver time grid. Receiver will calculate effective period of the time grid using the value of this parameter and values of relevant receiver options (see `/par/raw/curmsint` below). In turn, this time grid defines the rate of receiver generating pseudoranges, carrier phases and other measurements, as well as defines the base time grid for periodic messages output.

Effective Measurements Update Rate

Name: `/par/raw/curmsint`
Access: `r`
Type: `integer [milliseconds]`

Although the user can formally set `/par/raw/msint` to arbitrary allowed value, the receiver may need to adjust this user setting in order to make it consistent with the receiver options. The adjusted setting is stored to this read-only parameter and defines effective internal time grid.

The formula used to calculate `curmsint` is as follows:

$$\texttt{curmsint} = \mathit{max}\,(\texttt{msint}, \texttt{1000} \,/\, \mathit{max}\,(\texttt{1, \_RAW, \_POS, PDIF}))$$

where `_RAW`, `_POS`, and `PDIF` are current values of corresponding receiver options, and `msint` is the value of `/par/raw/msint` parameter.

The actual period at which receiver will allow user to get measurements depends on the value of `/par/raw/curmsint` parameter and the current value of receiver _RAW option. Actual measurements update period is calculated as follows:

$$\texttt{meas\_period} = \mathit{max}\,(\texttt{1, } \lfloor \texttt{1000} \,/\, \texttt{\_RAW} \,/\, \texttt{curmsint} \rfloor) \times \texttt{curmsint}$$

where `_RAW` is the current value of corresponding receiver option, and $\lfloor x \rfloor$ denotes integer part of x.

Note: While the formulas seem rather complex, what they basically mean in practice, is that if you set `/par/raw/msint` to a value that is multiple of `1000/_RAW`, then both the `/par/raw/curmsint` and actual allowed measurements output period will be equal to the specified value.

Pseudorange Smoothing Interval

Name: `/par/raw/smi`
Access: `rw`
Type: `integer [seconds]`
Values: `[0…900]`
Default: `300`

`0` – receiver will not use carrier phases to smooth pseudoranges. Therefore, in this case the pseudorange noise error will depend only on the corresponding DLL bandwidth (see the parameter `/par/raw/gdl/band` on page 76).

`[1…900]` – receiver will smooth pseudoranges based on a Kalman filter whose time constant is set to the value of this parameter.

Doppler Smoothing Interval

Name: `/par/raw/dopp/smi`
Access: `rw`
Type: `integer`
Values: `[0…2]`
Default: `2`

`0` – receiver outputs raw (non-smoothed) doppler that is instantaneous yet rather noisy.

`1` – doppler is computed using two consecutive carrier phase measurements. Such Doppler measurements are less noisy than in the first case.

`2` – doppler is computed using three consecutive carrier phase measurements. Doppler measurements obtained in this mode are least noisy.

Ionosphere Corrections Smoothing Interval

Name: `/par/raw/iono/smi`
Access: `rw`
Type: `integer [seconds]`
Values: `[0…900]`
Default: `60`

This parameter specifies the nominal interval (T_{nom}) over which raw ionospheric corrections are smoothed (assuming the receiver has been working for some time and has already obtained enough raw ionospheric corrections to perform such smoothing).

Note that the current ionosphere smoothing interval will vary in time. After you switch receiver on, the current smoothing interval will be growing from zero to the nominal value as new raw ionospheric corrections are computed. Once the nominal value is reached, this smoothing interval will be fixed. Smoothing filter is a simple n-point running average.

Minimum Ionosphere Corrections Smoothing Interval

Name: `/par/raw/iono/minsmi`
Access: `rw`
Type: `integer [seconds]`
Values: `[0…900]`
Default: `30`

The parameter specifies the minimum smoothing interval (T_{min}) for the receiver to filter raw ionospheric corrections before they can be used in position computation.

Multipath Reduction Parameters

CA/L1 Code Multipath Reduction

Name: `/par/raw/corr/ca/code`
Access: `rw`
Type: `enumerated`
Values: `normal,mpnew`
Default: `normal`

`normal` – CA/L1 code multipath reduction is off.

`mpnew` – CA/L1 code multipath reduction is on.

CA/L1 Carrier Phase Multipath Reduction

Name: `/par/raw/corr/ca/carrier`
Access: `rw`
Type: `enumerated`
Values: `normal,mpnew`
Default: `normal`

`normal` – CA/L1 carrier phase multipath reduction is off.

`mpnew` – CA/L1 carrier phase multipath reduction is on.

Tracking Loops Parameters

The user should distinguish between "guiding" and "guided" loops. In JAVAD GNSS receivers, C/A PLLs are referred to as "guiding loops" whereas the other lock loops (specifically, C/A DLLs, P/L1 PLLs, P/L1 DLLs, P/L2 PLLs, P/L2 DLLs) are called "guided loops".

C/A PLL Bandwidth

Name: `/par/raw/pll/band`
Access: `rw`
Type: `float [Hz]`
Values: `[20…50]`
Default: `25`

C/A Delay Loop Bandwidth

Name: `/par/raw/cagdl/band`
Access: `rw`
Type: `float [Hz]`
Values: `[0.1…50]`
Default: `0.8`

Care should be taken that the setting used doesn't result in receiver malfunction. Values in the range [0.2…5] Hz are usually safe.

What can prevent you from using more narrow bandwidths in guided loops? The main limitation here is ionosphere fluctuations and quick multipath.

Alternatively, what may prevent you from using a bandwidth larger than 5 Hz? The answer is GPS anti-spoofing, which results in much lower signal-to-noise ratios as compared to C/A measurements. Although GLONASS has no mechanism similar to GPS anti-spoofing, GLONASS channels are, however, also formally subject to these limitations since there are no separate settings for GPS and GLONASS loops in the receiver.

P/L1 and P/L2 Guided Loops Bandwidth

Name: `/par/raw/gdl/band`
Access: `rw`
Type: `float [Hz]`
Values: `[0.1…10]`
Default: `0.5`

This parameter governs all of the “guided” loops but the C/A DLLs, specifically, P/L1 PLLs, P/L1 DLLs, P/L2 PLLs and P/L2 DLLs. Care should be taken when using values other than the default (see notes to the previous parameter).

Guiding and Common Lock Loops Order

Name: `/par/raw/pll/order`
Access: `rw`
Type: `integer`
Values: `2,3`
Default: `3`

Care should be exercised when changing this parameter from the default value to “2”. We don't recommend you to use a 2nd order PLL unless you are certain that the receiver's trajectory is far from being a uniformly accelerated motion, and therefore using a lower-order PLL might make a difference. Imagine you are a field operator carrying a GPS/GLONASS receiver in your backpack and you need to collect raw data for your RTK project. In this and in similar cases, using a 2nd order PLL may pay off.

Enable Adaptive Guided Loops

Name: `/par/raw/gdl/adapt`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `on`

`on` – receiver will adjust the guided loops bandwidths depending on the actual strengths of the signals tracked. The weaker the signals, the narrower the bandwidths.

`off` – no adaptation will be performed.

P-code Loops Bandwidth Scale Factor

Name:	`/par/raw/pcll/scale`
Access:	`rw`
Type:	`float`
Values:	`[0.1…100]`
Default:	`2.0`

This parameter establishes the ratio between P-code PLL bandwidth and P-code CLL bandwidth.

P-code Loops Adaptation

Name:	`/par/raw/ploops/adapt`
Access:	`rw`
Type:	`float`
Values:	`[0.1…100]`
Default:	`99.0`

This parameter specifies the adaptation bandwidth of P-code loops.

Note: Non-default values of this parameter must be used for testing purposes only.

Common Loops Parameters

Common Loops Mode

Name:	`/par/raw/clp/loops`
Access:	`rw`
Type:	`boolean`
Values:	`on,off`
Default:	`off`

`on` – common loops mode is turned on.

`off` – common loops mode is turned off.

When common loops mode is turned on, you may want to change either or both of the common loops bandwidth and individual loops bandwidth.

Note: For more information about the *common loops* mode, refer to JAVAD GNSS website at *http://www.javad.com*.

Individual Loops Bandwidth

Name: `/par/raw/clp/indband`
Access: `rw`
Type: `float [Hz]`
Values: `[2…20]`
Default: `5`

Common Loops Bandwidth

Name: `/par/raw/clp/comband`
Access: `rw`
Type: `float [Hz]`
Values: `[2…50]`
Default: `25`

Receiver Dynamics for Common Loops

Name: `/par/raw/clp/static`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `off`

`on` – assume receiver doesn't move. Receiver will be able to acquire satellites with lower signal to noise ratios. This is achieved by reducing the number of unknowns estimated (specifically, from 4 down to 1), which is possible only for a static receiver.

`off` – don't assume receiver is static for the purposes of common loops.

Additional (L2-specific) Common Loop

Name: `/par/raw/clp2/loop`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `on`

`on` – receiver will use common loop to tracks L2 carrier phases.

`off` – receiver won't use common loop to track L2 carrier phases.

Note that in dynamic applications the measured carrier phase will also depend on antenna rotation around its axis. Abrupt antenna rotation can bring about significant additional carrier phase dynamics, which may result in poor L2 phase tracking.

Common PLL Order

Name: /par/raw/clp/pll/order
Access: rw
Type: integer
Values: 2,3
Default: 3

Measurements Extrapolation Time

Name: /par/raw/clp/wait
Access: rw
Type: integer [seconds]
Values: [1…100]
Default: 10

This parameter specifies time interval during which receiver will extrapolate satellite measurements should the loss of lock on the signal occur.

Anti-jamming Parameters

Anti-jamming Mode

Name: /par/ajm/mode
Access: rw
Type: enumerated
Values: on,off
Default: on

on – anti-jamming is turned on in the mode specified by the rest of parameters in this section.

off – anti-jamming is turned off.

Enable Anti-jamming Bands

Name: /par/ajm/band
Access: rw
Type: list {gps1,gps2,gps5,glo1,glo2} of boolean
Values: {on|off,on|off,on|off,on|off,on|off}
Default: {on,on,on,on,on}

on – enable anti-jamming on corresponding band.

off – disable anti-jamming on corresponding band.

Enable Anti-jamming on Band B.

Name: `/par/ajm/band/B (B=[gps1,gps2,gps5,glo1,glo2])`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `on`

`on` – enable anti-jamming on band `B`.

`off` – disable anti-jamming on band `B`.

Oscillator Parameters

Oscillator Frequency Offset Reduction Mode

Name: `/par/osc/mode`
Access: `rw`
Values: `off,locked,tied`
Default: `off`

`off` – receiver will not adjust internal oscillator’s frequency.

`locked` – receiver will adjust internal oscillator’s frequency until the measured frequency offset is reduced to about zero. By using the incoming satellite signals, the receiver will force the internal oscillator to generate a very stable output frequency signal. This frequency output is available via the corresponding receiver output pin.

`tied` – receiver will adjust both the internal oscillator’s frequency until the measured frequency offset is reduced to about zero and the internal clock until it gets synchronized with the reference time scale (e.g., GPS time).

After switching from `off` to `locked`, it may take the receiver up to a minute to adjust the internal oscillator frequency to the nominal value. This time is even longer for switching to `tied` mode.

After switching from `locked` or `tied` to `off`, the internal oscillator frequency will be rebound to its quiescent value abruptly, which may result in temporary loss of lock to satellites.

Setting this parameter to `locked` or `tied` only guarantees that the receiver's frequency output will have high long-term stability, not necessarily high short-term stability (the latter depends on the internal oscillator type). However, there is a way to guarantee that both of these characteristics are good even when using the internal crystal oscillator. It can be achieved by switching the receiver to common loops mode.

Antenna Input Parameters

Antenna Input

Name: `/par/ant/inp`
Access: `rw`
Type: `enumerated`
Values: `int,ext,auto (receiver-dependent)`
Default: `(receiver-dependent)`

Note that allowed parameter values and the default value are receiver-dependent.

Antenna Current Input

Name: `/par/ant/curinp`
Access: `r`
Type: `enumerated`
Values: `int,ext`

This parameter always reflects the actual antenna input that is currently in use. This is of primary interest when `/par/ant/inp` is set to `auto`.

Status of External Antenna

Name: `/par/ant/dc`
Access: `r`
Type: `enumerated`
Values: `off,normal,overload`

`off` – external antenna does not draw any DC

`normal` – external antenna draws normal DC

`overload` – external antenna draws too high DC

Frequency Source Parameters

Frequency Input

Name: `/par/frq/input`
Access: `rw`
Type: `enumerated`
Values: `int,ext`
Default: `int`

`int` – use internal oscillator as frequency source

`ext` – use external frequency input as frequency source

External Frequency Value

Name: /par/frq/ext
Access: rw
Type: integer [MHz]
Values: [2…40]
Default: 10

The frequency of the external oscillator. The receiver will not lock on satellites if this parameter is set to a wrong value while /par/frq/input is set to ext.

External Frequency Status

Name: /par/frq/stat
Access: r
Type: enumerated
Values: off,wait,locked

off – the receiver is using the internal oscillator.

wait – the receiver is waiting for the external frequency lock. The receiver will return this value if, after the user has set /par/frq/input to ext, the external frequency oscillator is disconnected, its amplitude is too low, or the actual external source frequency is different from that specified via /par/frq/ext parameter.

locked – external frequency source is being used.

External Frequency Amplitude

Name: /par/frq/amp
Access: r
Type: enumerated
Values: off,low,ok

off – the internal oscillator is used, – the amplitude can't be measured.

low – external frequency signal's amplitude is lower than required.

ok – external frequency signal's amplitude meets the required specifications.

3.4.5 Almanac Status

Almanac Status

Name: /par/alm
Access: r
Type: list {gps,glo}

Almanac Status for GPS Satellites

Name: /par/alm/gps
Access: r
Type: array [1…32] of boolean
Values: {y|n,…,y|n}

Almanac Status for GPS Satellite Number N

Name: /par/alm/gps/N (N=[1…32])
Access: r
Type: boolean
Values: y,n

y - the almanac data are available for the satellite

n - the almanac data are unavailable for the satellite

Almanac Status for GLONASS Satellites

Name: /par/alm/glo
Access: r
Type: array [1…24] of boolean
Values: {y|n,…,y|n}

Almanac Status for GLONASS Satellite Number N

Name: /par/alm/glo/N (N=[1…24])
Access: r
Type: boolean
Values: y,n

y - the almanac data are available for the satellite

n - the almanac data are unavailable for the satellite

3.4.6 Positioning Parameters

Generic Positioning Parameters

Position Update Rate

Name: /par/pos/msint
Access: rw
Type: integer [milliseconds]
Values: [50,100,150,…,5000]
Default: 100

This parameter specifies the required period of the position updates. Receiver will calculate effective period of the position updates using the value of this parameter and the value of the parameter `/par/raw/curmsint` (see `/par/pos/curmsint` below).

Effective Position Update Rate

Name: `/par/pos/curmsint`
Access: `r`
Type: `integer [milliseconds]`

Although the user can formally set `/par/pos/msint` to arbitrary allowed value, the receiver may need to adjust this user setting in order to make it consistent with the value of `/par/raw/curmsint` parameter. The adjusted setting is stored to this read-only parameter and defines internal effective position update rate.

The formula used to calculate `curmsint` is as follows:

$$\texttt{curmsint} = max\,(1, \lfloor \texttt{msint} / \texttt{raw} \rfloor) \times \texttt{raw}$$

where `msint` is the value of `/par/pos/msint`, `raw` is the value of `/par/raw/curmsint`, and $\lfloor x \rfloor$ denotes integer part of x.

The actual period at which receiver will allow user to get position depends on the value of `/par/pos/curmsint` parameter and the current value of corresponding receiver option (_POS for single point and code differential[2] position; and PDIF for carrier phase differential position). Actual position update period is calculated as follows:

$$\texttt{pos\_period} = max\,(1, \lfloor \texttt{1000} / \texttt{opt} / \texttt{curmsint} \rfloor) \times \texttt{curmsint}$$

where `opt` is the current value of corresponding receiver option, and $\lfloor x \rfloor$ denotes integer part of x.

Note: While the formulas seem rather complex, what they basically mean in practice, is that if you set `/par/pos/msint` to a value that is multiple of both `/par/raw/curmsint` and `1000/opt`, then both the `/par/pos/curmsint` and actual allowed position output period will be equal to the specified value.

Elevation Mask for Position Computation

Name: `/par/pos/elm`
Access: `rw`
Type: `integer [degrees]`
Values: `[-90…90]`
Default: `5`

Satellites with elevations lower than this mask will be excluded from position computation.

2. Code differential position will be available only if CDIF option is enabled.

SNR Mask for Position Computation

Name: `/par/pos/minsnr`
Access: `rw`
Type: `integer [dB*Hz]`
Values: `[0…50]`
Default: `30`

Satellites whose signal-to-noise ratios are lower than this value will be excluded from position computation.

Position Computation Mode

Name: `/par/pos/mode/cur`
Access: `rw`
Type: `enumerated`
Values: `pd,pf,cd,wd,sp`
Default: `sp`

`pd` – carrier phase differential (RTK) with fixed ambiguities

`pf` – carrier phase differential (RTK) with float ambiguities

`cd` – code differential (DGPS) mode

`wd` – wide area code differential mode (WDGPS) using SBAS corrections

`sp` – single point positioning mode[3]

These computation modes are arranged in the order of increasing of their typical accuracy, `sp` being the lowest, and `pd` being the highest accuracy mode. Receiver will try to compute position according to the mode specified by this parameter. If for whatever reason it fails to compute corresponding position, it will try to use the next mode below the current one, provided the mode is enabled by corresponding parameter (see below). This process continues down the list of modes until either some position is enabled and could be computed, or all the modes are exhausted.

Note: In fact receiver will still compute single point position for its internal purposes, though it will not make it available for output if disabled.

Note: Current implementation of code differential mode has a limitation. There are cases when receiver will not try to compute single point position when it failed to compute code differential position even though there are enough data.

3. Also known as “absolute positioning”, “stand-alone positioning” or simply “point positioning”

Enable Single Point Position

Name: `/par/pos/mode/sp`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `on`

`on` – when receiver is running in `pd`, `pf`, or `cd` mode and is unable to output `pd`, `pf`, or `cd` solution, it will output `sp` solution, if available.

`off` – receiver will not output `sp` solution unless it runs in `sp` mode.

This parameter doesn't affect behavior of receiver running in `sp` modes.

Enable Code Differential Position

Name: `/par/pos/mode/cd`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `off`

`on` – when receiver is running in `pd` or `pf` mode and is unable to output `pd` or `pf` solution, it will output `cd` solution, if available.

`off` – receiver will not output `cd` solution unless it runs in `cd` mode.

This parameter doesn't affect behavior of receiver running in `cd` or `sp` modes.

Note: Code differential mode requires broadcasting the corresponding DGPS (not RTK) messages from the reference receiver and accepting them on the rover receiver. If any of these requirements are not met, then enabling this parameter will not have any effect.

Enable RTK Solution with Float Ambiguities

Name: `/par/pos/mode/pf`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `on`

`on` – when receiver is running in `pd` mode and is unable to output `pd` solution, it will output `pf` solution, if available.

`off` – receiver will not output `pf` solution unless it is running in `pf` mode.

This parameter doesn't affect behavior of receiver running in `pf`, `cd`, or `sp` modes.

Enable Satellite System

Name: `/par/pos/sys`
Access: `rw`
Type: `list {gps:boolean,glo:boolean,gal:boolean}`
Values: `{on|off,on|off,on|off}`
Default: `{on,on,on}`

This parameter allows you to select satellite constellation(s) used for position computation. The first and the second fields correspond to GPS, GLONASS, and GALILEO, respectively.

Enable GPS Satellites by Their Numbers

Name: `/par/pos/gps/sat`
Access: `rw`
Type: `array [1…32] of boolean`
Values: `{y|n,…,y|n}`
Default: `{y,…,y}`

Enable GPS Satellite Number N

Name: `/par/pos/gps/sat/N (N=[1…32])`
Access: `rw`
Type: `boolean`
Values: `y,n`
Default: `y`

`y` – enable using of GPS satellite number `N` for position computation

`n` – disable using of GPS satellite number `N` for position computation

Enable GLONASS Satellites by Their Numbers

Name: `/par/pos/glo/sat`
Access: `rw`
Type: `array [1…24] of boolean`
Values: `{y|n,…,y|n}`
Default: `{y,…,y}`

Enable GLONASS Satellite Number N

Name: `/par/pos/glo/sat/N (N=[1…24])`
Access: `rw`
Type: `boolean`
Values: `y,n`
Default: `y`

- `y` – enable using of GLONASS satellite with orbit number `N` for position computation
- `n` – disable using of GLONASS satellite with orbit number `N` for position computation

If given satellite is enabled by this parameter, has frequency code number `M`, and is disabled by parameter `/par/pos/glo/fcn/M`, the satellite will be still disabled.

Enable GLONASS Satellites by FCN

Name: `/par/pos/glo/fcn`
Access: `rw`
Type: `array [-7…12] of boolean`
Values: `{y|n,…,y|n}`
Default: `{y,…,y}`

Enable GLONASS Satellite with FCN #N

Name: `/par/pos/glo/fcn/N (N=[-7…12])`
Access: `rw`
Type: `boolean`
Values: `y,n`
Default: `y`

- `y` – enable using of GLONASS satellite with frequency code number `N` for position computation
- `n` – disable using of GLONASS satellite with frequency code number `N` for position computation

If given satellite is enabled by this parameter, has orbit number `M`, and is disabled by parameter `/par/pos/glo/sat/M`, the satellite will be still disabled.

Enable GALILEO Satellites by Their Numbers

Name: `/par/pos/gal/sat`
Access: `rw`
Type: `array [1…32] of boolean`
Values: `{y|n,…,y|n}`
Default: `{y,…,y}`

Enable GALILEO Satellite Number N

Name: `/par/pos/gal/sat/N (N=[1…32])`
Access: `rw`
Type: `boolean`
Values: `y,n`
Default: `y`

`y` – enable using of GALILEO satellite number `N` for position computation

`n` – disable using of GALILEO satellite number `N` for position computation

Reference Time Scale

Name: `/par/pos/reftime`
Access: `rw`
Type: `enumerated`
Values: `gps,glo,gal`
Default: `gps`

The time scale of corresponding system becomes the reference time scale for position computation purposes, i.e. receiver time offset and other systems time offsets are computed relative to that reference system time.

`gps` – use GPS system time as reference time scale

`glo` – use GLONASS system time as reference time scale

`gal` – use GALILEO system time as reference time scale

Datums

Notations

Each datum supported by the receiver has unique datum identifier assigned. Datum identifier is a string of up to 5 upper-case characters. We will use the type `datum_id` to refer datum identifiers. Though the set of supported datums may vary from receiver to receiver and from one firmware version to another, the following datums are always supported:

`W84` – WGS-84 datum; GPS system datum

`P90` – PE-90 datum; GLONASS system datum

`W72` – WGS-72 datum

`USER` – user-defined datum

In addition, receiver may support a subset of datums described in the *"Reference Ellipsoids and Local Datums supported by JAVAD GNSS Receivers"* guide.

Note: You can get the list of supported datums along with their parameters from receiver itself using `print,/par/pos/datum:on` command.

Every datum has corresponding ellipsoid parameters as well as a set of parameters for standard 7-parameters transformation. For most datums these parameters are read-only. User may change only parameters of `USER` and `P90` datum.

Ellipsoid parameters have type called `ell_params` of the following format:

`{ell_id,axis,inv_flat}`

where

`ell_id` - ellipsoid identifier. String comprising two characters.

`axis` - ellipsoid's major semi-axis in the range `[6300000…6500000]` meters.

`inv_flat` - ellipsoid's inverse flattening in the range `[280…300]`, dimensionless.

Set of parameters for 7-parameters transformation have type called `datum_params` of the following format:

`{datum_id,ref,dx,dy,dz,rx,ry,rz,scale}`

where

`datum_id` - datum identifier (see above).

`ref` - flag indicating whether the datum's transformation parameters are specified with respect to WGS-84 (`ref=0`) or PE-90 (`ref=1`)

`dx, dy, dz` - translations in X-, Y- and Z-direction, respectively. Each component is in the range `[-10000…10000]` meters.

`rx, ry, rz` - rotations around X-, Y- and Z-axis, respectively. Each component is in the range `[-60…60]` seconds of arc.

`scale` - scale in ppm (true scale = `scale` x 10^{-6}) ranging within `[-100…100]`.

The above parameters specify a coordinate transformation from given datum to WGS-84 (or PE-90) according to the following equations:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix}_{W84|P90} = \begin{bmatrix} \Delta X \\ \Delta Y \\ \Delta Z \end{bmatrix} + (1 + s \cdot 10^{-6}) \cdot \begin{bmatrix} 1 & R_z & -R_y \\ -R_z & 1 & R_x \\ -R_y & -R_x & 1 \end{bmatrix} \cdot \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}_{local}$$

Finally, in the descriptions below, `[D]` denotes either one of the valid datum identifiers, or the string `INUSE` that designates the datum that is set as current for position computation.

Current Datum for Position Computation

Name: /par/pos/datum/cur
Access: rw
Type: datum_id
Default: W84

This parameter specifies the identifier of the datum that will be used for position computation. Note that some of the receiver messages always contain position referenced to WGS-84 datum. This parameter has no effect on such messages.

Use Datum Rotations for Position Computation

Name: /par/pos/datum/ft
Access: rw
Type: boolean
Values: on,off
Default: off

on – receiver will apply the full transformation matrix, including rotations.

off – receiver will not apply rotations, i.e., computations are made as if all the rotations are zero.

JAVAD GNSS recommends that you set this parameter to on only when the rotation parameters used to relate the orientation of datums are big enough (tens seconds of arc or more). In all other cases, leave the default parameter's value (off).

Parameters of Datum [D]

Name: /par/pos/datum/[D]
Access: r
Type: list {ell,datum}

ell – reference ellipsoid parameters for this datum

par – 7-parameters transformation for this datum

Reference Ellipsoid for Datum [D]

Name: /par/pos/datum/[D]/ell
Access: r
Type: ell_params

7-parameters Transformation for Datum [D]

Name: /par/pos/datum/[D]/par
Access: r
Type: datum_params

User Defined Datum

Name: `/par/pos/datum/USER`
Access: `rw`
Type: `list {ell,par}`
Default:
```
{{U-,6378137.0000,298.257223563},
 {USER,0,+0.0000,+0.0000,+0.0000,
  +0.00000,+0.00000,+0.00000,+0.00000}}
```

`ell` – user-defined ellipsoid parameters for this datum. This should conform to the format of the type `ell_params`. The `ell_id` field of the format should be set to the string `U-` (the letter "U" in the upper case and the "minus" sign), or, alternatively, could be omitted (note that the delimiting comma should still exist).

`par` – user-defined set of parameters for 7-parameters coordinates transformation. This should conform to the format of type `datum_params`. The `datum_id` field of the format should be set to the string `USER`, or, alternatively, could be omitted (note that the delimiting comma should still exist).

Example: Set parameters of user-defined datum, then print them:

```
⇒ set,/par/pos/datum/USER,{{U-,6378136,298},{USER,0,0,0,1,0,0,-0.2,0}}
⇒ print,/par/pos/datum/USER
⇐ RE021{{U-,6378136.0000,298.000000000},
  RE046 {USER,0,+0.0000,+0.0000,+1.0000,+0.00000,+0.00000,
    -0.20000,+0.00000}}
```

7-parameters Transformation for PE-90

Name: `/par/pos/datum/P90/par`
Access: `rw`
Type: `datum_params`
Default:
```
{P90,0,+0.0000,+0.0000,+1.0000,
 +0.00000,+0.00000,-0.20626,+0.00000}
```

The `datum_id` field of the format should be set to the string `P90`, or, alternatively, could be omitted (note that the delimiting comma should still exist).

The `ref` field of the format should be set to the value `0`, or, alternatively, could be omitted (note that the delimiting comma should still exist).

There are no universally accepted Helmert transformation parameters for the PE-90 datum so far. This explains why the user is allowed to define his/her own version of the transformation using this parameter. The default value for this parameter was taken from *"Parameters of the Earth 1990 (PZ-90). KNITs, 1998"* (in Russian).

Note: Under use of combined GPS/GLONASS receivers in DGPS modes, be sure that the same transformation parameters for PE-90 datum is used at both the base station and the rover. You may

relax this requirement provided referencing corrections to local datum is turned on (see `/par/rtcm/base/locdtm` parameter on page 173).

Grid Systems

Current Grid System

Name: `/par/pos/grid/cur`
Access: `rw`
Type: `enumerated`
Values: `NONE,UTM,USER,LOC`
Default: `UTM`

`NONE` – receiver will not compute grid coordinates (this mode allows some reduction of the processor's computation load).

`UTM` – Universal Transverse Mercator (automatic selection of the right zone). 6-degree zones with the scale = 0.9996.

`USER` – user-defined grid system.

`LOC` – local coordinates (as a result of the localization procedure).

Note that computation of grid and local coordinates depends on the parameter `/par/pos/datum/cur`. Ensure that this parameter is specified properly.

User-defined Grid System

Name: `/par/pos/grid/USER`
Access: `rw`
Type: `{proj,lat0,lon0,scale,falseN,falseE}`
Default: `{TM,N00d00m00.000000s,E000d00m00.000000s, 1.000000000,0.0000,0.0000}`

`proj` – map projection identifier, either `TM` for Transverse Mercator projection, or `STER` for Stereographic projection.

`lat0` – latitude of the origin of the grid projection, in latitude format.

`lon0` – longitude of the central meridian of the projection, in longitude format.

`scale` – scale factor; `[0.1…10]`.

`falseN` – false Northing; $[-10^7 \ldots 10^7]$ meters.

`falseE` – false Easting; $[-10^7 \ldots 10^7]$ meters.

Specific Map Projection

Name: `/par/pos/grid/spc`
Access: `rw`
Type: `enumerated`
Values: `NONE,S34J,S34S,S34B`
Default: `NONE`

There are map projections that require additional computations, for example, with the use of polynomials. This parameter allows the user to select such a specific map projection. Currently the receiver supports three such projections which are used on Denmark maps: System 34 Jutland (`S34J`), System 34 Seeland (`S34S`), and System 34 Bornholm (`S34B`).

Note: For correct use of a projection from this list, you should set up the parameters of the desired Transverse Mercator projection using the `/par/pos/grid/USER` parameter and specify the correct datum transformation parameters using the `/par/pos/datum/USER` parameter.

Local Coordinates

Parameters of Transformation to Local Coordinates

Name: `/par/pos/local/par`
Access: `rw`
Type: `{lat0,lon0,scalePrj,falseN,falseE,delN,delE,`
`scaleLoc,rotation,Hx,Hy,Ho}`
Default: `{N00d00m00.000000s,E000d00m00.000000s,1,0,0,0,0,`
`1,000d00m00.000000s,0,0,0}`

Stereographic projection parameters:

`lat0` – latitude of the origin of the projection, in latitude format.

`lon0` – longitude of the origin of the projection, in longitude format.

`scalePrj` – scale factor of the projection; `[0.1…10]`.

`falseN` – false Northing; $[-10^7 \ldots 10^7]$ meters.

`falseE` – false Easting; $[-10^7 \ldots 10^7]$ meters.

Parameters obtained from the localization procedure:

`delN` – offset in North direction; $[-10^7 \ldots 10^7]$ meters.

`delE` – offset in East direction; $[-10^7 \ldots 10^7]$ meters.

`scaleLoc` – scale factor; `[0.1…10]`.

`rotation` – rotation angle; `[0…360)` degrees.

`Hx, Hy, Ho` – parameters that relate to the height transformation; `Hx` and `Hy` are dimensionless, [-1…1]; `Ho` is in range $[-10^4 \ldots 10^4]$ meters.

$$
\begin{aligned}
n &= scale \cdot (x_{stereo} \cdot \cos\alpha + y_{stereo} \cdot \sin\alpha) + \Delta N \\
e &= scale \cdot (x_{stereo} \cdot \sin\alpha + y_{stereo} \cdot \cos\alpha) + \Delta E \\
h &= H_o + H_x \cdot x_{stereo} + H_y \cdot y_{stereo}
\end{aligned}
$$

Generic Single Point Parameters

Parameters described in this section affect single point positioning. Please note that due to specific of code differential (DGPS) positioning, most of these parameters will affect DGPS positioning as well.

Measurements Type to Use

Name: `/par/pos/sp/meas`
Access: `rw`
Type: `enumerated`
Values: `ca,p1,p2,l2c,l5,ionofree,all`
Default: `ca`

This parameter specifies which measurements receiver will use for single point position computation.

`ca` – use CA/L1 code measurements

`p1` – use P/L1 code measurements

`p2` – use P/L2 code measurements

`l2c` – use CA/L2 code measurements

`l5` – use L5 code measurements

`ionofree` – use ionosphere-free combination of code measurements

`all` – use all the available signals. In this case optimal combination of signals is used for every SV

Enable Ionospheric Corrections

Name: `/par/pos/sp/iono`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `on`

`on` – receiver will correct the measured pseudoranges for ionospheric delay errors before computing the point position. For the ionospheric model used, please refer to ICD-GPS-200C, Revision IRN-200C-004 April 12, 2000.

`off` – receiver will not use ionospheric corrections.

Note: Ionospheric corrections are used in the receiver exclusively for computing single point position. Receiver messages will contain raw pseudoranges.

Enable Tropospheric Corrections

Name: `/par/pos/sp/tropo`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `on`

`on` – receiver will correct the measured pseudoranges for tropospheric delay errors when computing the point position. For the tropospheric model applied, please refer to “Technical characteristics of the NAVSTAR GPS” (June 1991).

`off` – receiver will not use tropospheric corrections.

Note: Tropospheric corrections are used in the receiver exclusively for computing single point position. Receiver messages will contain raw pseudoranges.

PDOP Mask

Name: `/par/pos/pdop`
Access: `rw`
Type: `float`
Values: `[0…500]`
Default: `30`

If the current PDOP value exceeds the specified mask, the receiver will not compute the single point or code differential position.

Local Time Zone

Name: `/par/pos/ltz`
Access: `rw`
Type: `list {integer,integer}`
Values: `{[-13…+13],[0…+59]}`
Default: `{0,00}`

The first parameter in the list describes the local zone hour offset from the UTC time. The second parameter in the list describes the local zone minute offset from the UTC time. These local zone hour and minute offsets will be used in NMEA ZDA message.

Fixed Geoidal Separation

Name: `/par/pos/geoidh`
Access: `rw`
Type: `float [meters]`
Values: `[-1000…1000]`
Default: `0`

This parameter allows to enter user-defined geoidal separation. If the parameter `/par/pos/fix/geoidh` is set to `on`, the user-defined geoidal separation value is used for orthometric height computation (overriding its counterpart from the in-built geoidal model if this model is enabled in the receiver).

Use Fixed Geoidal Separation

Name: `/par/pos/fix/geoidh`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `off`

JAVAD GNSS receivers incorporate the global geoid model from "Technical characteristics of the NAVSTAR GPS" (June 1991), which is available if the receiver option _GEO is enabled.

If the receiver option _GEO is enabled, setting the parameter `/par/pos/fix/geoidh` to `on` will instruct the receiver to use an alternative geoidal separation from the parameter `/par/pos/geoidh` for the geoidal separation value computed through the built-in model[4].

Positioning With Reduced State Vector

The parameters described in this section provide positioning modes with fixed values of some state vector entries. These modes are useful when the values of some states are known from some external source or from previous positioning epoch. (The conventional example of that kind is 2D positioning with fixed altitude.) These modes of positioning sometimes can improve position accuracy, decrease required number of SVs, and/or increase SVs redundancy.

4. As a rule, this is only applicable to static receivers.

Fixed Altitude Positioning

Entered Altitude

Name: `/par/pos/alt`
Access: `rw`
Type: `float [meters]`
Values: `[-1000…10000]`
Default: `0`

This parameter allows to enter exact ellipsoidal height of the antenna phase center in the currently used datum. When `/par/pos/fix/alt` is set to `on`, this value will be used in position computations decreasing the number of parameters to be calculated.

This parameter serves two purposes:

- Using an a priori ellipsoidal height will allow the receiver to get a position fix in critical situations when there are few satellites in sight and when it is impossible to derive the point solution using the current measurements only.
- Using precise a priori ellipsoidal height estimate allows to have more precise position fixes.

Use Fixed Altitude

Name: `/par/pos/fix/alt`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `off`

`on` – enable receiver to use in position computation the fixed ellipsoidal height specified by the `/par/pos/alt` parameter.

`off` – receiver will calculate altitude.

Fixed Receiver Clock Drift Positioning

Entered Clock Drift

Name: `/par/pos/clkdft`
Access: `rw`
Type: `float [m/s]`
Values: `[-10000…10000] or string "last"`
Default: `0`

This parameter allows to enter exact value of clock drift parameter. When `/par/pos/fix/clkdft` is set to `on`, this value will be used in position computations (for

extrapolation of last clock offset value) decreasing the number of parameters to be calculated.

The string `last` entered instead of numerical value will assign the numberical value of the last computed clock drift value to this parameter.

This parameter serves two purposes:

- Using an a priori clock drift will allow the receiver to get a position fix in critical situations when there are few satellites in sight and when it is impossible to derive the point solution using the current measurements only.
- Using precise a priori clock drift estimate allows to have more precise position fixes.

Note that using of this parameters makes sence only for operation modes utilizing stable external reference oscillator.

Use Fixed Clock Drift

Name: `/par/pos/fix/clkdft`
Access: `rw`
Type: `boolean`
Values: `off,on`
Default: `off`

`on` – enable receiver to use in position computation the fixed clock drift (extrapolated clock offset) specified by the `/par/pos/clkdft` parameter.

`off` – receiver will calculate clock drift.

Fixed GPS to GLONASS Time Offset Positioning

Fixed GPS to GLONASS Time Offset

Name: `/par/pos/gpsglo`
Access: `rw`
Type: `float [meters]`
Values: `[-300000…+300000] or string "last"`
Default: `0`

This parameter determines the a priori known (constant) time offset between the GLONASS and GPS time scales. Note that this offset is entered in meters, not in time units (just divide this value by the speed of light to get the offset in seconds).

The string `last` entered instead of numerical value will assign the numberical value of the last computed time offset value to this parameter.

This parameter serves two purposes:

- Using a priori time offset will allow the receiver to get a position fix in critical situations when there are few satellites in sight and when it is impossible to derive the point solution using the current measurements only. E.g. if there are only three GPS satellites and one GLONASS satellite in view, the receiver won't be able to get a position fix unless the user enters a GLONASS vs. GPS time offset or some other a priori data, thus reducing the number of unknowns in the corresponding set of equations.
- Using precise a priori time offset will allow you to have more precise position fixes.

Use Fixed GPS to GLONASS Time Offset

Name: `/par/pos/fix/gpsglo`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `off`

`on` - receiver will use constant time offset specified by `/par/pos/gpsglo` in position computation.

`off` - receiver will calculate inter-system time offset.

Fixed GPS to GALILEO Time Offset Positioning

Fixed GPS to GALILEO Time Offset

Name: `/par/pos/gpsgal`
Access: `rw`
Type: `float [meters]`
Values: `[-300000…+300000] or string "last"`
Default: `0`

This parameter determines the a priori known (constant) time offset between the GALILEO and GPS time scales. Note that this offset is entered in meters, not in time units (just divide this value by the speed of light to get the offset in seconds).

The string `last` entered instead of numerical value will assign the numberical value of the last computed time offset value to this parameter.

This parameter serves two purposes:

- Using a priori time offset will allow the receiver to get a position fix in critical situations when there are few satellites in sight and when it is impossible to derive the point solution using the current measurements only. E.g. if there are only three GPS satellites and one GALILEO satellite in view, the receiver won't be able to get a position fix unless the user enters a GALILEO vs. GPS time off-

set or some other a priori data, thus reducing the number of unknowns in the corresponding set of equations.

- Using precise a priori time offset will allow you to have more precise position fixes.

Use Fixed GPS to GALILEO Time Offset

Name: `/par/pos/fix/gpsgal`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `off`

`on` – receiver will use constant time offset specified by `/par/pos/gpsgal` in position computation.

`off` – receiver will calculate inter-system time offset.

Held Parameters Positioning

The parameters in this section allow receiver to hold last computed values of some elements of the state vector when number of SVs is not enough to perform calculations with all the elements considered as unknown.

Enable to Hold Altitude

Name: `/par/pos/hold/alt`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `on`

`on` – receiver can hold last computed altitude for position computation in the case when SVs number is not enough for conventional position computation.

`off` – receiver is not allowed to hold last computed altitude.

Enable to Hold Clock Drift

Name: `/par/pos/hold/clkdft`
Access: `rw`
Type: `boolean`
Values: `off,on`
Default: `off`

`on` – receiver can hold last computed clock drift (extrapolate last computed clock offset) for position computation in the case when SVs number is not enough for conventional position computation.

`off` – receiver is not allowed to hold last computed clock drift.

Enable to Hold Intersystem Time Offsets

Name: `/par/pos/hold/systime`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `on`

`on` – receiver can hold last computed intersystem time scales offsets for position computation in the case when SVs number is not enough for conventional position computation.

`off` – receiver is not allowed to hold last computed time offsets.

KFK Parameters

KFK is positioning engine based on Kalman filter. It could be used in standalone or code differential positioning modes with or without WAAS/EGNOS. KFK provides sufficient increase of positioning accuracy, availability, continuity, and integrity, especially:

- for mobile dynamic user
- in case of unfavorable environment conditions (dense canopy, city canyon, etc.)

KFK Mode

Name: `/par/pos/kfk/mode`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `on`

`on` – use KFK engine for stand-alone and code-differential positioning modes.

`off` – use conventional least squares (LMS) based engine for stand-alone and code-differential positioning modes.

This parameter has no effect when receiver is in one of RTK positioning modes, i.e., when `/par/pos/mode/cur` parameter is set to either `pf` or `pd`. In this case the LMS engine is used to compute stand-alone or code-differential solution.

KFK Reset

Name: `/par/pos/kfk/reset`
Access: `w`
Type: `boolean`
Values: `on`

Setting this pseudo-parameter to `on` will initialize the KFK engine from conventional least squares based positioning engine.

KFK Dynamics Mode

Name:	`/par/pos/kfk/dynamic/mode`
Access:	`rw`
Type:	`enumerated`
Values:	`static,walk,car,ship,aircraft,unlim,satellite,user,adapt`
Default:	`car`

The dynamic mode for KFK is defined by the values of horizontal and vertical components of dynamic noise variance and maximum velocity.

`static,walk,car,ship,aircraft,satellite,unlim` - use one of preset dynamic modes from the table below.

`user` - use the mode specified by the /par/pos/kfk/dynamic/user parameter.

`adapt` - use adaptive mode for dynamic parameters.

Table 3-1. Preset KFK Dynamic Modes

Mode	var(V_h) m^2/s^2	var(V_v) m^2/s^2	max(V_h) m/s	max(V_v) m/s
static	0.0001	0.0001	0	0
walk	1	1	5	5
car	10	1	70	20
ship	50	5	40	5
aircraft	100	100	500	100
satellite	10	10	20000	20000
unlim	10000	10000	5000	5000

User Defined KFK Dynamics

Name:	`/par/pos/kfk/dynamic/user`
Access:	`rw`
Type:	`list {VarVh:float,VarVv:float,` `MaxVh:float,MaxVv:float}`
Values:	`{[0.0001…10000],[0.0001…10000],[0…5000],[0…5000]}`
Default:	`{1.0,1.0,5.0,5.0}`

Using this parameter, the user can set KFK dynamics parameters according to his specific requirements. The values from this parameter take effect only when parameter `/par/pos/kfk/dynamic/mode` is set to the value `user`.

RAIM Parameters

RAIM Mode

Name: `/par/pos/raim/mode`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `on`

`on` – RAIM is active

`off` – RAIM is turned off

Alarm Limit Mode

Name: `/par/pos/raim/al/mode`
Access: `rw`
Type: `enumerated`
Values: `manual,npa,ter,enr`
Default: `npa`

In the description of this parameter, *nmi* stands for International Nautical Mile, that is equal to 1852 meters.

`npa` – non-precision approach. Limit is equal to 0.3nmi.

`ter` – terminal. Limit is equal to 1.0nmi.

`enr` – en route. Limit is equal to 2.0nmi.

`manual` – RAIM will use the contents of the parameter `/par/pos/raim/al/manual` as the alarm limit.

Alarm Limit for Manual Mode

Name: `/par/pos/raim/al/manual`
Access: `rw`
Type: `float [meters]`
Values: `[10…10000]`
Default: `555.6 (it corresponds to "npa" mode);`

This parameter specifies alarm limit value for the `manual` alarm limit mode. The default value of this parameter numerically corresponds to the value used in `npa` mode.

Filtering Position Estimates

Position Filter Mode

Name: /par/pos/filt/mode
Access: rw
Type: boolean
Values: on,off
Default: off

Position Filter Type

Name: /par/pos/filt/type
Access: r
Type: enumerated
Values: stat

stat – simple N-point weighted moving average is used to smooth the current position.

Note that this type of position filter normally applies only to static (or "nearly static") receivers in sp or cd mode. It is especially useful when the number of tracked satellites changes abruptly; in this case the position accuracy may temporarily drop dramatically unless the position filter is on. For moving receivers, using this type of position filter may adversely affect the receiver trajectory's accuracy.

Position Filter Width

Name: /par/pos/filt/num
Access: rw
Type: integer
Values: [1…10000]
Default: 30

This parameter designates the number of the preceding pre-filtered position measurements used by the least-squares estimator to obtain the current smooth position. Provided /par/pos/filt/type is stat (the only one currently supported), this estimator is just an N-point weighted running average.

Maximum Allowed Time Gap

Name: /par/pos/filt/maxdel
Access: rw
Type: integer [seconds]
Values: [1…3600]
Default: 10

This parameter specifies the maximum allowed time gap mask for two successive position estimates. If the current position estimate is obtained in more than maximum allowed time gap seconds after the previous one, the position filter is reset.

Reset the Position Filter

Name: `/par/pos/filt/reset`
Access: `rw`
Type: `boolean`
Values: `on,off`

`on` – the position filter will be reset. Immediately after the reset, the parameter will automatically be set back to `off`.

`off` – ignored.

Position Filter Weighting Mode

Name: `/par/pos/filt/weight`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `off`

`off` – position filter will treat all of the position estimates as uniformly weighted measurements.

`on` – each position estimate will have its own weight depending on the corresponding *rms* error.

3.4.7 Timing Parameters

Improved Timing Mode

Running receiver in the Improved Timing mode serves two main purposes:

- It enables you to synchronize your receiver with the GPS and/or GLONASS time scales even when you have only one GPS satellite and/or one GLONASS satellite in sight.
- This mode provides better synchronization precision as compared to the general case when the receiver has to solve for both coordinates and time offsets.

To correctly use the Improved Timing mode, you need to specify reference coordinates of the receiver antenna L1 phase center as precise as possible. Refer to “Reference Parameters” on page 143 for details. Note that 1 meter of error in reference position will result in about 3.3 nanoseconds error in time offset.

Enable Improved Timing Mode

Name: `/par/pos/clk/fixpos`
Access: `rw`
Type: `boolean`
Values: `on, off`
Default: `off`

`on` – receiver will use the a priori known coordinates from the `/par/ref/syspos/gps` and `/par/ref/syspos/glo` parameters to solve for the unknown time offsets between corresponding system time scale and receiver time.

`off` – receiver will calculate the time offsets as part of usual position computation.

Pulse Per Second (PPS) Parameters

Overview

JAVAD GNSS receivers can generate precise Pulse Per Second (PPS) signals with programmable reference time system, period and offset. PPS signals are available via the corresponding output connector pins (for this information, please see the JAVAD GNSS receiver pinout description at the JAVAD GNSS website). These trigger signals may be synchronized to the reference time grid in two ways.

You can either synchronize the falling edge of the PPS signal with the specified reference time or synchronize the rising edge of this signal with the reference time.

The PPS time grid is defined by the PPS `period` as follows:

$$\text{reference\_time} \; mod \; \texttt{period} = 0$$

In addition PPS *offset* allows you to generate PPS shifted with respect to the PPS time grid. Positive values of *offset* means shift to the future.

There could be up to two PPS outputs in JAVAD GNSS receivers, “a” (PPSA) and “b” (PPSB). It is possible to use both PPS outputs concurrently.

Due to a hardware limitation, PPS signals are discrete with a resolution of 25 ns. JAVAD GNSS receivers, however, allow compensation for this “discreteness error”. You can force the receiver to generate for each PPS pulse a message containing the offset between the scheduled PPS time and the actual pulse edge's arrival time. Refer to “[ZA], [ZB] PPS Offset” on page 313 for details.

Parameters

In this section, the notation `[a|b]` denotes either PPSA or PPSB. The user should substitute either `a` or `b`.

Enable PPS Generation

Name: `/par/dev/pps/[a|b]/out`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `on`

`on` – corresponding PPS will be generated

`off` – corresponding PPS is disabled

PPS Reference Time

Name: `/par/dev/pps/[a|b]/time`
Access: `rw`
Type: `enumerated`
Values: `gps,glo,utcusno,utcsu`
Default: `utcusno`

`gps` – GPS system time

`glo` – GLONASS system time

`utcusno` – UTC(USNO) time scale

`utcsu` – UTC(SU) time scale

When this parameter is set to `gps` or `utcusno`, there should be at least one GPS satellite being locked for receiver to be able to synchronize to corresponding time scale. Similarly, when this parameter is set to `glo` or `utcusno`, there should be at least one GLONASS satellite being locked.

In static applications where the receiver's precise position is known, we recommend that you switch your receiver to the Improved Timing mode. Refer to “Improved Timing Mode” on page 106 for details.

Tie PPS to its Reference Time

Name: `/par/dev/pps/[a|b]/tied`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `on`

There are applications where the user needs to synchronize PPS signals with either the receiver’s internal clock or an external frequency, not with the selected reference time. This parameter allows not to tie PPS to its reference time.

`on` – PPS pulses are synchronized with the selected reference time, which is the common practice.

`off` – PPS signals are synchronized either with the receiver’s internal clock or, provided the parameter `/par/frq/input` has been set to `ext`, with the external frequency.

PPS Period

Name: `/par/dev/pps/[a|b]/per/ms`
Access: `rw`
Type: `integer [milliseconds]`
Values: [10…10^9]
Default: `1000`

Milliseconds of PPS Offset

Name: `/par/dev/pps/[a|b]/offs/ms`
Access: `rw`
Type: `integer [milliseconds]`
Values: [-10^9/2…10^9/2]
Default: `0`

Nanoseconds of PPS Offset

Name: `/par/dev/pps/[a|b]/offs/ns`
Access: `rw`
Type: `integer [nanoseconds]`
Values: [-10^6/2…10^6/2]
Default: `0`

PPS Reference Edge

Name: `/par/dev/pps/[a|b]/edge`
Access: `rw`
Type: `enumerated`
Values: `rise,fall`
Default: `rise`

`rise` – rising edge of the PPS pulse will be tied to the reference time

`fall` – falling edge of the PPS pulse will be tied to the reference time

Period of Marked PPS Pulses

Name: `/par/dev/pps/[a|b]/mper`
Access: `rw`
Type: `integer [milliseconds]`
Values: 0,[20…10^9]
Default: `0`

The JAVAD GNSS receiver can generate either or both normal and "marked" PPS pulses. This parameter specifies the period of the marked PPS signal. Note that the width of marked pulse is about 1.5 times greater than the width of normal PPS pulse.

`0` – receiver will generate no marked pulses.

`NM=[20…10`^{9}`]` – provided the parameter governing the period of normal pulses is set to `NN`, then the receiver will generate both normal and marked pulses, but marked pulses will be output only every `N` milliseconds, where `N` is equal to the least common multiple of `NN` and `NM`.

External Event Parameters

Overview

JAVAD GNSS receivers have the "event marking" function allowing the user to measure/record event times in the specified reference time system with high accuracy. You may have your JAVAD GNSS receiver measure the time of either the rising edge or falling edge of the input event signal. Most of the JAVAD GNSS receivers may accommodate up to two external event pins, EventA and EventB.

The measured event times are buffered[5] inside the receiver and could be then output by the corresponding receiver message(s), please see "[XA], [XB] External Event" on page 313 for details.

Parameters

In this section, the notation `[a|b]` denotes either EventA or EventB. The user should substitute either `a` or `b`.

Enable Event Acquisition

Name:	`/par/dev/event/[a|b]/in`
Access:	`rw`
Type:	`boolean`
Values:	`on,off`
Default:	`off`

`on` – corresponding event input is active and events will be acquired and buffered.

`off` – corresponding event input will be inactive.

5. The internal buffer will hold up to 128 most recent events.

Event Reference Time

Name: `/par/dev/event/[a|b]/time`
Access: `rw`
Type: `enumerated`
Values: `gps, glo, utcusno, utcsu`
Default: `utcusno`

`gps` – GPS system time

`glo` – GLONASS system time

`utcusno` – UTC(USNO) time scale

`utcsu` – UTC(SU) time scale

When this parameter is set to `gps` or `utcusno`, there should be at least one GPS satellite being locked for receiver to be able to synchronize to corresponding time scale. Similarly, when this parameter is set to `glo` or `utcusno`, there should be at least one GLONASS satellite being locked.

In static applications where the receiver's precise position is known, we recommend that you switch your receiver to the Improved Timing mode. Refer to “Improved Timing Mode” on page 106 for details.

Tie Measured Event Time to its Reference Time

Name: `/par/dev/event/[a|b]/tied`
Access: `rw`
Type: `enumerated`
Values: `on,off`
Default: `on`

With this parameter, the receiver is instructed to measure the event reception time in the selected reference time with or without consideration for the computed receiver clock offset.

`off` – the event time is measured in the receiver time scale that will differ from the selected reference time by the computed clock offset.

`on` – the event time is measured in the selected reference time properly. Thus the name of the parameter, tied (figuratively speaking, we “tie up” event signals rigidly with the selected reference time).

Event Reference Edge

Name: `/par/dev/event/[a|b]/edge`
Access: `rw`
Type: `enumerated`
Values: `rise,fall`
Default: `rise`

`rise` – the time of the rising edge of the event signal will be measured

`fall` – the time of the falling edge of the event signal will be measured

Synchronize Receiver Clock with External Event

Name: `/par/dev/event/[a|b]/lock`
Access: `rw`
Type: `enumerated`
Values: `on,off`
Default: `off`

`on` – receiver will synchronize its one-millisecond cycle grid with the corresponding edge of the next event signal arrived after the setting this parameter to `on`. You may need to set this parameter to `on` repeatedly to ensure that the receiver maintains synchronization with the time scale governing the external event signals.

`off` – receiver will not synchronize its clock with external event.

Status of the Receiver Clock Synchronization

Name: `/par/dev/event/[a|b]/locked`
Access: `r`
Type: `boolean`
Values: `on,off`

`on` – receiver time has been synchronized with an external event.

`off` – receiver time is not synchronized with external event.

Current Time

Parameters described in this section hold current time that is updated every millisecond. To receive consistent values of multiple parameters, use single `print` command to retrieve multiple values, for example:

Example: Get consistent UTC time and date:

```
⇒ print,/par/time/utc
⇐ RE019{2006-12-26,14:43:11.269}
```

Example: Get snapshot of all the times along with their names:

```
⇒ print,/par/time:on
⇐ RE018/par/time={rcv=53091337,
  RE02A utc={date=2006-12-26,clock=14:44:37.337},
  RE01D gps={week=383,ms=225891337},
  RE01C glo={day=1091,ms=63877337}}
```

Current Times

Name: `/par/time`
Access: `r`
Type: `list {rcv,utc,gps,glo}`

`rcv` – receiver time

`utc` – UTC time

`gps` – GPS system time

`glo` – GLONASS system time

Current Receiver Time

Name: `/par/time/rcv`
Access: `r`
Type: `integer [milliseconds]`
Values: `[0…86400000)`

This parameter reports the current time in local (i.e., receiver) time scale.

Current UTC Time

Name: `/par/time/utc`
Access: `r`
Type: `list {date,clock}`

`date` – UTC date

`clock` – UTC clock (time of day)

Current UTC Date

Name: `/par/time/utc/date`
Access: `r`
Type: `string`

This parameter reports the current UTC date and this date is represented as `YYYY-MM-DD`, where:

`YYYY` – the year in the Gregorian calendar between 0001 and 9999

`MM` – the month of the year between 01 (January) and 12 (December)

`DD` – the day of the month between 01 and 31. If no time information is available, the receiver reports an empty string.

Current UTC Time of Day

Name: `/par/time/utc/clock`
Access: `r`
Type: `string`

This parameter reports the current time of day in UTC as a value in the `HH:MM:SS.SSS` or empty string format, depending on whether the time information is available or not.

`HH` – hours [00-23]

`MM` – minutes [00-59]

`SS` – seconds [00-60][6]

`SSS` – milliseconds [000-999]

Current GPS Time

Name: `/par/time/gps`
Access: `r`
Type: `list {week,ms}`

`week` – GPS week number

`ms` – GPS time inside week

Current GPS Week

Name: `/par/time/gps/week`
Access: `r`
Type: `integer`
Values: `[-1…1023]`

This parameter reports the current GPS week number. If no time information is available, the receiver reports -1.

Current GPS Time of Week

Name: `/par/time/gps/ms`
Access: `r`
Type: `integer [milliseconds]`
Values: `[-1…604800000)`

6. It could be equal to 60 only when UTC leap second happens.

This parameter reports the current GPS time of week. If no time information is available, the receiver reports `-1`.

Current GLONASS Time

Name: `/par/time/glo`
Access: `r`
Type: `list {day,ms}`

`day` – GLONASS day number

`ms` – GLONASS time inside day

Current GLONASS Day Number

Name: `/par/time/glo/day`
Access: `r`
Type: `integer`
Values: `[-1,1…1461]`

This parameter reports the current GLONASS day number within the 4-year period beginning with the leap year. If no time information is available, the receiver reports `-1`.

Current GLONASS time of day

Name: `/par/time/glo/ms`
Access: `r`
Type: `integer [milliseconds]`
Values: `[-1…86400000)`

This parameter reports the current time of day in GLONASS system time. If no time information is available, the receiver reports `-1`.

3.4.8 Code Differential (DGPS) Parameters

Generic DGPS Parameters

Source Port of DGPS Corrections

Name: `/par/pos/cd/port`
Access: `rw`
Type: `enumerated`
Values: `any,/[port]`
Default: `any`

`any` – receiver will use differential corrections from whichever port.

`/[port]` – receiver will only use differential corrections from corresponding port.

Note that when there are multiple active sources of DGPS correction, you either need to choose only one of them through this parameter, or turn on the multi-base DGPS mode, otherwise receiver may fail to calculate correct position due to mixing of differential corrections from different sources.

Maximum Age of DGPS Corrections

Name: `/par/pos/cd/maxage`
Access: `rw`
Type: `integer [seconds]`
Values: `[1…1200]`
Default: `30`

Receiver will stop using differential corrections for DGPS solution when their age exceeds specified limit.

Ionosphere-free DGPS Mode

Name: `/par/pos/cd/ionofree`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `off`

`on` – receiver will use both ionosphere corrections from RTCM message type 15 and “regular” differential corrections from RTCM message types 1 and 31 or 9 and 34 when computing position.

`off` – receiver will not use ionosphere corrections from RTCM message type 15.

Maximum Age of Ionosphere Corrections

Name: `/par/pos/cd/iono/maxage`
Access: `rw`
Type: `integer [seconds]`
Values: `[1…1800]`
Default: `300`

Receiver will stop using ionosphere corrections for DGPS solution when their age exceeds specified limit.

Range Residual Limit

Name: `/par/pos/cd/rlim`
Access: `rw`
Type: `float [meters]`
Values: `[1.0…100.0]`
Default: `5.0`

Satellites whose range residuals are greater than this limit will be discarded from code differential positioning. This parameter is used only if the parameter `/par/pos/raim/mode` has been set to `on`.

Source of DGPS Corrections

Name: `/par/pos/cd/src/mode`
Access: `rw`
Type: `enumerated`
Values: `user,best,nearest,any`
Default: `nearest`

`user` – receiver will use corrections with station ID specified by the parameter `/par/pos/cd/src/usersrc`.

`nearest` – receiver will use corrections from the nearest reference station.

`best` – receiver will use reference station with minimal estimated RMS of navigation solution. Navigation satellites which don't have corrections from this source won't be used in position solution.

`any` – receiver will use reference station with minimal estimated RMS of navigation solution. Navigation satellites which don't have corrections from this source will get corrections from another source with larger estimated RMS, if possible.

Fixed Reference Station ID

Name: `/par/pos/cd/src/usersrc`
Access: `rw`
Type: `integer`
Values: `[0…1023]`
Default: `0`

This parameter specifies user defined source of corrections by reference station ID. Receiver will only use corrections from refernce station with given ID, provided `/par/pos/cd/src/mode` is set to `user`.

Corrections to Reference Stations Coordinates

Name: `/par/rover/base/pos/par/X (X=[a…e])`[7]
Access: `rw`
Type: `{valid,port,basedID,delNorth,delEast,delUp}`

`valid` – boolean [`on|off`]. If `valid` is `off`, the parameter is associated with no reference station.

7. Currently GREIS provides only five parameters for coordinate offsets since only up to five reference stations can be used in multi-base mode.

`port` – this field takes same values as, for example, the parameter `/par/pos/cd/port`. It specifies the port from which the coordinates to be corrected are coming.

`baseID` – integer [1…1023]. It is identification of the reference station for which the offsets are specified. If `baseID` is `-1`, the offsets are applicable to all reference stations used in multi-base mode except the stations for which offsets are already specified with the preceding parameters (see the example below).

`delNorth, delEast, delUp` – float numbers specifying the north-, east- and up-components of the vector correction to the reference station position in meters.

Each specification from `a` to `e` in turn is compared to the port and identifier of the reference station. The first specification that has `valid` field set to `on` and which `port` and `baseId` match those of the reference station, provides offsets for the coordinates of this reference station. If no specifications match, the offsets are assumed to be zero.

You can use these parameters to compensate for known “coordinate offsets” of up to five reference stations used in multi-base code differential mode. This correction mechanism is especially important when differential corrections from different reference stations are used together to compute the rover position in the mixed solution mode (for details, see parameter `/par/pos/cd/src/mode` on page 117). Should the transmitted reference coordinates of the base stations be conspicuously different from their “truth” coordinates in this mode, the estimated rover position may prove corrupt unless the user enters appropriate “coordinate offsets” for these stations on the rover side.

Example:

```
⇒ set,rover/base/pos/par/a,{on,/dev/ser/a,any,-0.02,0.033,-0.05}
⇒ set,rover/base/pos/par/b,{on,any,101,0.01,-0.034,-0.22}
⇒ set,rover/base/pos/par/c,{on,any,102,0.002,0.023,-0.011}
```

With these commands, the receiver will apply the offset (-0.02,0.03,-0.05) to the transmitted coordinates of the reference station whose differential corrections are received on serial port A (whatever the station ID). It will apply corresponding corrections to reference stations with IDs 101 and 102, unless data for them are coming from serial port A, in which case the first offsets apply.

SBAS (WAAS/EGNOS) Parameters

SBAS stands for Satellite Based Augmentation System. JAVAD GNSS receivers support two implementations of SBAS, Wide Area Augmentation System (WAAS), and European Geostationary Navigation Overlay Service (EGNOS). Parameters described in this section define when and how SBAS data will be used by the receiver to increase positioning accuracy.

SBAS Mode

Name: `/par/pos/wd/mode`
Access: `rw`
Type: `enumerated`
Values: `manual,npa,ter,enr`
Default: `manual`

This parameter defines the mode of use of SBAS corrections for DGPS solution.

`manual` – receiver will apply SBAS corrections according to other parameters defined in this section.

`npa,ter,enr` – receiver will apply SBAS corrections according to the DO-229C/D specification.

Enable SBAS Mode for Non-safety Applications

Name: `/par/pos/wd/nsa`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `on`

`on` – receiver will apply SBAS corrections as for non-safety application. Currently receiver will interpret non-empty message type 0 as message type 2 in accordance with DO-229C/D. Note that this mode is used only when parameter `/par/pos/wd/mode` is set to `manual`.

`off` – receiver will apply SBAS corrections as for safety applications.

SBAS Elevation Mask

Name: `/par/pos/wd/elm`
Access: `rw`
Type: `integer`
Values: `[-90…90]`
Default: `5`

SBAS corrections from satellites with elevations lower than this mask will be excluded from position computation.

Enable SBAS Corrections by Satellite Numbers

Name: `/par/pos/wd/sat`
Access: `rw`
Type: `array [120…138] of boolean`
Values: `{y|n,…,y|n}`
Default: `{y,…,y}`

Enable SBAS Corrections from Satellite Number N

Name: `/par/pos/wd/sat/N (N=[120…138])`
Access: `rw`
Type: `boolean`
Values: `y,n`
Default: `y`

`y` – enable using of SBAS satellite number N as source of corrections

`n` – disable using of SBAS satellite number N as source of corrections

Source of SBAS Corrections

Name: `/par/pos/wd/src/mode`
Access: `rw`
Type: `enumerated`
Values: `user,best,any`
Default: `any`

`user` – enable using of SBAS satellite with PRN specified by the parameter `/par/pos/wd/src/usersrc`.

`best` – receiver will choose SBAS satellite with minimal estimated RMS of navigation solution as source. Navigation satellites which don't have corrections from this source are not used in position solution.

`any` – receiver will choose SBAS satellite with minimal estimated RMS of navigation solution as source. Navigation satellites which don't have corrections from this source will get corrections from another source with larger estimated RMS if possible.

Note: This parameter takes effect only when `/par/pos/wd/mode` is set to `manual`.

Fixed SBAS Satellite

Name: `/par/pos/wd/src/usersrc`
Access: `rw`
Type: `integer`
Values: `[120…138]`
Default: `120`

Specify SBAS satellite number as user-defined source of corrections. This parameter is active only when `/par/pos/wd/src/mode` parameter is set to `user`.

Note: The SVs specified by this parameter should also be enabled by the `/par/pos/wd/sat` parameter in order to be used as source of corrections.

Enable SBAS Ionosphere Corrections

Name: `/par/pos/wd/ionofree`
Access: `rw`
Type: `boolean`
Values: `off,on`
Default: `off`

`on` – receiver will use both ionosphere corrections and satellite (fast and long term) corrections from SBAS satellites for DGPS solution.

`off` – receiver will not use ionosphere corrections from SBAS satellites for DGPS solution.

Maximum Age of SBAS Satellite Corrections

Name: `/par/pos/wd/maxage`
Access: `rw`
Type: `integer [seconds]`
Values: `[1…3600]`
Default: `360`

Receiver will stop using SBAS satellite corrections for DGPS solution when their age exceeds specified limit.

Note: This limit is active for SBAS fast corrections as well as for SBAS long-term corrections.

Note: This parameter takes effect only when `/par/pos/wd/mode` is set to `manual`.

Maximum Age of SBAS Ionosphere Corrections

Name: `/par/pos/wd/iono/maxage`
Access: `rw`
Type: `integer [seconds]`
Values: `[1…3600]`
Default: `1200`

Receiver will stop using SBAS ionosphere corrections for DGPS solution when their age exceeds specified limit.

Note: This parameter takes effect only when `/par/pos/wd/mode` is set to `manual`.

Smoothing Interval of SBAS Satellite Corrections

Name: `/par/pos/wd/smi`
Access: `rw`
Type: `integer [seconds]`
Values: `[0…3600]`
Default: `60`

Receiver will smooth SBAS satellite corrections (the sum of SBAS fast corrections and SBAS long-term corrections) before their use for DGPS solution. This parameter defines the value of time constant of the smoothing filter. Zero value stops the smoothing.

Note: This parameter takes effect only when `/par/pos/wd/mode` is set to `manual`.

Smoothing Interval of SBAS Ionosphere Corrections

Name:	`/par/pos/wd/iono/smi`
Access:	`rw`
Type:	`integer [seconds]`
Values:	`[0…3600]`
Default:	`60`

Receiver will smooth SBAS ionosphere corrections before their use for DGPS solution. This parameter defines the value of time constant of the smoothing filter. Zero value stops the smoothing.

Note: This parameter takes effect only when `/par/pos/wd/mode` is set to `manual`.

3.4.9 Phase Differential (RTK) Parameters

Generic RTK Parameters

RTK Position Computation Mode

Name:	`/par/pos/pd/mode`
Access:	`rw`
Type:	`enumerated`
Values:	`extrap,delay`
Default:	`delay`

`extrap` – in this mode the RTK engine will extrapolate the latest carrier phases received from reference station to the current time. The final positioning accuracy may be somewhat lower due to additional extrapolation errors, which may be up to a few millimeters vertical and horizontal for a one-second extrapolation time. Note that this mode could be used only when reference station is static (i.e., not moving).

`delay` – in this mode, the RTK engine does not extrapolate the base station's carrier phases in position computation. Instead, the engine will either compute a delayed position or simply output the current stand-alone position (while waiting for new differential messages from the base station). Note that the delayed position is computed for the time (epoch) to which the last received base station's carrier phase measurements correspond. Accuracies achievable in delay mode are normally on a level with those of post-processing kinematic.

RTK Delay Mode Variant

Name: `/par/pos/pd/delay`
Access: `rw`
Type: `enumerated`
Values: `last,every`
Default: `last`

This parameter is only active for RTK *delay* mode.

`last` – RTK engine will process the last set of carrier phase differential data received from the reference station.

`every` – RTK engine will attempt to process all sets of carrier phase differential data sequentially received from the reference station.

Update Interval of RTK Reference Station

Name: `/par/pos/pd/period`
Access: `rw`
Type: `float [seconds]`
Values: `[0.05…100]`
Default: `1`

This parameter is effective only in the RTK *delay* mode. Its value should be set to the exact rate at which the base station transmits its differential correction data. This parameter will instruct the rover receiver to output the RTK position at the same rate at which differential corrections are updated.

Extrapolate Missing Carrier Phase Measurements

Name: `/par/pos/pd/se`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `on`

This parameter is only active in RTK *delay* mode.

`on` – RTK engine will extrapolate the missing carrier phase measurements for the currently processed epoch (provided that the code phase measurements have been successfully received for this epoch).

`off` – RTK won't extrapolate missing carrier phase measurements.

Maximum Time Gap in Reference Data

Name: `/par/pos/pd/timegap`
Access: `rw`
Type: `integer [seconds]`
Values: `[-1…3600]`
Default: `-1`

`[0…3600]` – RTK will be reset as soon as duration of time gap in the data being received from reference station exceeds specified value.

`-1` – RTK is never reset due to time gaps .

Source Port of Differential Data

Name: `/par/pos/pd/port`
Access: `rw`
Type: `enumerated`
Values: `any,/[port]`
Default: `any`

`any` – RTK engine will use differential data from whichever port.

`/[port]` – RTK engine will only use differential data received on the corresponding port.

Confidence Level for Ambiguity Fixing

Name: `/par/pos/pd/aflevel`
Access: `rw`
Type: `enumerated`
Values: `low,medium,high`
Default: `medium`

`low` – 95% confidence level

`medium` – 99.5% confidence level

`high` – 99.9% confidence level

The higher the confidence level specified, the longer the integer ambiguity search time and the higher the reliability of the ambiguity fixed solution.

Known Point Initialization

Name: `/par/pos/pd/fixpos`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `off`

`on` – RTK engine will use the rover's precise position for ambiguities resolution. This allows the engine to fix ambiguities much faster. The precise coordinates of the L1 phase center of the rover antenna must be specified as described in "Reference Parameters" on page 143. Care should be taken that this parameter is set back to `off` once the RTK initialization is over and the antenna starts moving. Otherwise the rover's position will be computed incorrectly.

`off` – RTK engine won't use rover precise position.

RTK Penalty Parameter

Name: `/par/pos/pd/pen`
Access: `rw`
Type: `float`
Values: `[0…1000]`
Default: `20`

The penalty parameter is used for the known point initialization function.

Use CA/L1 Measurements for RTK

Name: `/par/pos/pd/meas/ca`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `on`

Use P/L1 Measurements for RTK

Name: `/par/pos/pd/meas/p1`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `on`

Use P/L2 Measurements for RTK

Name: `/par/pos/pd/meas/p2`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `on`

Use L1 Only for RTK

Name: `/par/pos/pd/l1only`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `off`

This parameter allows the receiver to stabilize the fixed position in case of poor L2 phase tracking.

`on` – RTK engine will calculate the final position using only L1 measurements even if the ambiguities have been resolved for L2 as well.

`off` – RTK will use L2 if possible.

RTK Weighting Scheme

Name: `/par/pos/pd/scale`
Access: `rw`
Type: `integer`
Values: `0,1,2`
Default: `1`

This parameter specifies the weights that the engine will apply to single-differenced (SD) carrier phase measurements.

`0` – the simplest weighting scheme will be used. Specifically, all SD carrier phases will be used with the a priori weight *(1 /* `sigma`*2)*, where `sigma` is 0.05 cycles. This scheme, which is recommended when running RTK in favorable environment conditions, allows a shorter ambiguity fixing time than the other two (see below).

`1` – the first adaptive weighting scheme will be used. Specifically, SD carrier phases for the given satellite will be used with the a priori weight:

max(0.2, sin(`elev`*)) /* `sigma_est`*2)*

where `elev` designates the satellite's elevation angle, and `sigma_est` designates the RMS single-differenced carrier phase error.

`2` – the second adaptive weighting scheme will be used. This is similar to scheme 1, but different estimator to get `sigma_est` is used. This scheme is strongly recommended in scenarios where strong multipath is expected.

Enable RTK Ionosphere Model

Name: `/par/pos/pd/ion`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `on`

`on` – single-differenced ionosphere model is enabled

`off` – single-differenced ionosphere model is disabled

Threshold for RTK Ionosphere Model

Name: `/par/pos/pd/ionr`
Access: `rw`
Type: `float [meters]`
Values: `[0…106)`
Default: `4000`

The ionospheric delay will be modeled by RTK engine only if the estimated baseline length is greater or equal to the specified value.

Memory Factor For the Float Ambiguity Filter

Name: `/par/pos/pd/mem`
Access: `rw`
Type: `float`
Values: `[0.5…1.0]`
Default: `0.99970`

The smaller the filter memory factor, the "less important" are the older ambiguity estimates for the RTK engine estimating the current ambiguities. On shorter vectors (up to 8 km), it is recommended to set the memory factor to 0.998 when running the receiver under tree canopy.

Half-integer Ambiguity Fixing on L1 and L2

Name: `/par/pos/pd/si`
Access: `rw`
Type: `list {L1, L2} of boolean`
Values: `{on|off,on|off}`
Default: `{off,off}`

This is a technology (specialist) parameter. It is mostly used for internal JAVAD GNSS purposes when testing/debugging new firmware versions.

Interval of Verification of Fixed Ambiguities

Name: `/par/pos/pd/check`
Access: `rw`
Type: `integer`
Values: `[-1…32767]`
Default: `17`

This parameter specifies the periodicity N of the engine checking fixed ambiguities for errors. The engine will be forced to recompute/check the ambiguity vector every N epochs. Thus estimated “forced ambiguities” will be compared against the current ones. If the forced ambiguity vector differs from the current ambiguity vector, the float solution mode will be enabled at once.

`-1` – ambiguity verification is off.

`0, 1` – RTK engine will verify ambiguities every epoch.

`[2…32767]` – RTK engine will verify ambiguities every specified number of epochs.

Rover Dynamics for RTK

Name: `/par/pos/pd/dyn`
Access: `rw`
Type: `float`
Values: `[0…1]`
Default: `1`

`[0…1)` – RTK engine will run in static mode.

`1` – RTK engine will run in kinematic mode.

When in static mode, the engine uses a running average over a few consecutive raw estimates to decrease the resulting position's noise error.

Note that the RTK engine also provides a “static watchdog mechanism”. When in static mode, the receiver will automatically monitor estimated coordinates X, Y, Z. Should the position change over 4 centimeters in one of the coordinates, the receiver will immediately switch to kinematic mode and will run in this mode for the next 30 seconds.

RTK Computational Scheme Version

Name: `/par/pos/pd/ver`
Access: `r`
Type: `integer`
Values: `2`

Reset RTK Engine

Name: `/par/pos/pd/reset`
Access: `w`
Type: `boolean`
Values: `on,off`
Default: `off`

`on` – RTK engine will be reset and the value of this parameter will be set back to `off`.

`off` – ignored.

Enable Use of Kept Reference Coordinates

Name: `/par/pos/pd/ref/keep`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `off`

`on` – receiver will use reference coordinates retrieved from NVRAM at receiver startup for RTK.

`off` – receiver will not begin RTK processing prior to receiving reference coordinates from base station even if the rover has already received all the other necessary data/measurements from the base. Note that reference coordinates are normally transmitted much rarer than measurements and such delays may well be unacceptable for many applications.

Care should be taken when setting this parameter to `on`. Imagine for a moment that the rover has moved to a different location and started a new RTK session with a different reference station. Should this parameter be set to `on`, the rover receiver will be misusing the old reference coordinates for some time, which will most likely result in position blunders until the rover receives a first message with the correct reference coordinates.

Factor for Residual Ionosphere Standard Deviation

Name: `/par/pos/pd/ionf`
Access: `rw`
Type: `float`
Values: `[0…106]`
Default: `4`

RTK uses the following equation for the residual ionosphere standard deviation (measured in meters):

$$\text{stdev\_iono\_delay} = \texttt{ionf} * 10^{-6} * \text{base\_line\_length}$$

where `ionf` is the value of this parameter.

Environmental Condition Factor

Name: `/par/pos/pd/env`
Access: `rw`
Type: `enumerated`
Values: `open,forest`
Default: `open`

`open` – RTK will use “normal” thresholds when searching for measurement outliers. This setting is used if the environment conditions are considered favorable for RTK (many satellites in sight, few obstructions and low multipath).

`forest` – RTK will use less rigid thresholds when filtering out measurement outliers. This mode is recommended when working under tree canopy or in other cases of high multipath.

Use Smoothed Pseudorange in RTK

Name: `/par/pos/pd/smr`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `off`

`on` – RTK engine will use smoothed rover pseudorange measurements.

`off` – RTK engine will use raw rover pseudorange measurements.

RTK Maximum Extrapolation Time

Name: `/par/pos/pd/textr`
Access: `rw`
Type: `float`
Values: `[0…60]`
Default: `30`

When RTK works in the extrapolation mode and the time gap between the last received correction and the current rover's time exceeds this value, RTK stops to produce a position.

Maximum Number of Iterations for Float Ambiguity

Name: `/par/pos/pd/maxit`
Access: `rw`
Type: `integer`
Values: `[0…10]`
Default: `1`

This parameter specifies maximum number of iterations RTK will make when estimating float ambiguity.

Maximum Number of Iterations for Fixed Ambiguity

Name: `/par/pos/pd/maxitf`
Access: `rw`
Type: `integer`
Values: `[0…10]`
Default: `1`

This parameter specifies maximum number of iterations RTK will make when estimating residuals of fixed ambiguity.

Use FKP Data From RTCM 2.x Message 59

Name: `/par/pos/pd/fkp`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `off`

`on` – instructs the rover receiver to use ionospheric and geometric corrections from RTCM 2.x messages type 59 (FKP) when computing position.

`off` – RTK engine will ignore FKP data.

Maximum Number of Satellites to Use in RTK

Name: `/par/pos/pd/maxsat`
Access: `rw`
Type: `integer`
Values: `[4…20]`
Default: `20`

With this parameter, the user specifies maximum number of satellites that are used in RTK position computation. If the actual number of satellites in sight exceeds the current parameter's value, the RTK engine will utilize data only from the satellites with the highest CA/L1 SNRs and the number of satellites used will not be greater than that specified by the parameter.

Period of Base Measurements for Extrapolation

Name: `/par/pos/pd/experiod`
Access: `rw`
Type: `float [seconds]`
Values: `[0…5]`
Default: `0.1`

The RTK engine will extrapolate the received reference station's carrier phase measurements if the time to which these carrier phase measurements correspond is divisible by the value of this parameter.

Use Base Doppler in RTK

Name: `/par/pos/pd/usebasedop`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `on`

`on` – enable receiver to use the doppler measurements, either received from the base station or obtained by the base measurements extrapolator, in RTK processing.

`off` – disable the use of doppler measurements.

Period of Ambiguities Estimation

Name: `/par/pos/pd/afperiod`
Access: `rw`
Type: `float [seconds]`
Values: `[0.05…5]`
Default: `1`

Enable Measurements Quality Indicators

Name: `/par/pos/pd/qcheck`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `on`

`on` – RTK engine will take into account measurements quality indicators.

`off` – RTK will ignore measurements quality indicators.

RTK VRS Mode

Name: `/par/pos/pd/vrs`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `off`

`on` – RTK will assume that the reference station in use is Virtual Reference Station (VRS)

`off` – RTK will assume that the reference station is real one.

Correlation Time for Estimating the Residual Ionosphere

Name: `/par/pos/pd/iont`
Access: `rw`
Type: `float [seconds]`
Values: `[0.0…1800.0]`
Default: `0.0`

Minimum CA/L1 SNR

Name: `/par/pos/pd/minpot`
Access: `rw`
Type: `integer [dB*Hz]`
Values: `[1…60]`
Default: `31`

Only those satellites will be used in RTK position computation whose CA/L1 signal-to-noise ratios exceed the specified threshold value.

RMS for Pseudorange Measurements Noise

Name: `/par/pos/pd/rmsc`
Access: `rw`
Type: `float [meters]`
Values: `[0.001…100.0]`
Default: `5.0`

RMS for Carrier Phase Measurements Noise

Name: `/par/pos/pd/rmsp`
Access: `rw`
Type: `float [cycles]`
Values: `[0.001…1.0]`
Default: `0.05`

Maximum Distance Between Base and Rover

Name: `/par/pos/pd/range`
Access: `rw`
Type: `float [meters]`
Values: `[0.0…1000000.0]`
Default: `1000000.0`

With this parameter, the user specifies the maximum allowed distance between the reference and rover stations. If this distance exceeds the specified limit, the rover receiver will stop to provide the RTK position.

Minimum Number of Satellites for Fixing Integer Ambiguities

Name: `/par/pos/pd/minsat`
Access: `rw`
Type: `integer`
Values: `[4…20]`
Default: `5`

RTK Heading Parameters

Heading Mode

Name: `/par/pos/pd/hd/mode`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `off`

`on` – this value indicates that the mutual distance between the base and the rover antennas remain fixed during the RTK session. This restriction allows the RTK engine to fix ambiguities faster and more accurately.

Use Fixed Baseline Length

Name: `/par/pos/pd/hd/uselen`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `off`

`on` – RTK engine will use in the heading mode the a priori baseline length from the parameter `/par/pos/pd/hd/len/0`, unless its value is equal to zero.

`off` – RTK engine will compute its own baseline length estimate, which is obtained by averaging instantaneous baseline estimates over a 30-second interval after the first ambiguity fix. Thus, a derived empirical estimate is then used by the RTK engine to improve ambiguity fixing. Also note that this empirical estimate is constantly refined by the receiver as new measurements arrive.

Fixed Baseline Length

Name: `/par/pos/pd/hd/len/N (N=[0…2])`
Access: `rw`
Type: `float [meters]`
Values: `[0…10000]`
Default: `0`

Currently only parameter with N=0 is used by RTK engine. Unless the value of this parameter is zero, it will be used by RTK engine in the heading mode, provided `/par/pos/pd/hd/uselen` is set to `on`.

Penalty Factor for Baseline Length Term

Name: `/par/pos/pd/hd/pen`
Access: `rw`
Type: `float`
Values: `[0…10]`
Default: `0.25`

The larger this weight, the more "critical" for correct ambiguity resolution is the a priori baseline length specified by the parameter `/par/pos/pd/hd/len`.

The penalty factor must be consistent with the actual accuracy of the specified a priori baseline length. It it is not the case, the RTK engine may not be able to fix ambiguities correctly. In most scenarios, it will be sufficient to use the default value of the penalty factor.

The heuristic formula for the penalty factor is the following:

$$pf = 6 / \mathtt{sigma}2$$

where `sigma` (measured in millimeters) stands for the expected standard deviation of the specified baseline length. If the baseline length is a priori known with an accuracy of 1mm or better, the user is recommended to set the parameter to the maximum value 10.

Memory Factor for Smoothing of Heading Angles

Name: `/par/pos/pd/hd/mem`
Access: `rw`
Type: `float`
Values: `[0…1]`
Default: `0`

The bigger the memory factor value, the more conservative is smoothing.

TDMA Multiple Reference Stations

JAVAD GNSS receivers support Multi-base mode in which the rover receiver is able to obtain RTK data from more than one reference station[8]. Running in this mode, the reference stations broadcast their RTK data using TDMA (Time Division Multiple Access) method. This method allows the reference stations to use a single frequency for trans-

8. Currently, this mode allows the rover to use up to four reference stations.

mitting their data. It is achieved by setting a transmission delay for each station. This mode is currently supported for CMR Plus messages only.

Timeout for Data From Multiple Reference Stations

Name: `/par/pos/pd/mbtimeout`
Access: `rw`
Type: `integer`
Values: `[1…6]`
Default: `3`

In RTK with multiple reference stations, this parameter specifies how many epochs the RTK engine will wait for a complete data set, for example, from three reference stations before processing, for example, the data from only two of them.

Reference Station ID to Use

Name: `/par/pos/pd/inuse`
Access: `rw`
Type: `integer`
Values: `[-1…31]`
Default: `-1`

This parameter allows the user to identify the reference station (by specifying its CMR reference station ID) he/she wants to use in order to compute the RTK position.

`-1` – rover receiver will use RTK corrections from whichever station.

`[0…31]` – receiver will use RTK corrections only from the reference station having the specified ID. Data from all the other reference stations will be ignored. Thereby it guarantees that the rover will work properly if there are two or more reference stations transmitting RTK data on the same frequency.

Setting the parameter to `-1` whereas two or more sources of RTK data are available simultaneously and automatic selection of the nearest reference station is turned off, may result in inability to get the RTK solution since RTK data received from several reference stations may be mixed with each other.

Enable Automatic Selection of the Nearest Reference Station

Name: `/par/pos/pd/nrs/mode`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `off`

`on` – rover receiver will use RTK data broadcast by the nearest reference station. The receiver will use this parameter only if `/par/pos/pd/inuse` is set to `-1`.

`off` – rover receiver will use RTK data from any reference station.

Threshold for Switching of Nearest Reference Station

Name: `/par/pos/pd/nrs/lim`
Access: `rw`
Type: `float [meters]`
Values: `[1.0….10000.0]`
Default: `25.0`

Switching to the new reference station will occur only if the difference between the distance to the new reference station and the distance to the current reference station exceeds the specified limit.

Maximum counter of attempts to stay the nearest reference station

Name: `/par/pos/pd/nrs/cnt`
Access: `rw`
Type: `integer`
Values: `[1…30000]`
Default: `10`

The parameter defines the counter that serves for setting up a total number of attempts during which a nearest reference station must remain the nearest one for this station to be selected as the new nearest reference station. If during those attempts, at least one of the other reference stations is detected as the nearest one, the counter will be reset and started again. The rover estimates the distances to the reference stations and makes attempts to select the nearest one each time when it receives CMR message Type 0.

Attitude Parameters

Attitude Mode

Name: `/par/att/mode`
Access: `rw`
Type: `integer`
Values: `0,1`
Default: `0`

`0` – attitude mode is set off.

`1` – attitude mode is on.

Number of Epochs to Use for Self-calibration

Name: `/par/att/naver`
Access: `rw`
Type: `integer`
Values: `[1…2000000]`
Default: `60`

The number of measurement epochs must be set taking into consideration the following:

1. Minimum self-calibration time is 1 hour;
2. Recommended self-calibration time is greater than 1 hour;
3. Maximum self-calibration time is 2 hours.

According to the above, the number of measurement epochs can be determined as:

$$N = T_{calibration} / T_{interval}$$

where

$T_{calibration}$ – self-calibration time,

$T_{interval}$ – differential corrections update period.

For example, if differential correction update period is 0.05 s (1/20 Hz) and $T_{calibration}$ is 1 hour, then

$$N = 3600 / 0.05 = 72000$$

Start Self-calibration

Name: `/par/att/tune`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `off`

`on` – the self-calibration process will begin. The value of this parameter is immediately restored to `off`.

`off` – ignored.

Base Line Vectors of the Body Frame

Name: `/par/att/bl/N (N=[0,1,2])`
Access: `rw`
Type: `{n,r,b}`
Default: `{0,0,0}`

`n, r, b` – Nose-, Right-, Belly- components of the corresponding vector. Float values in the range `[-100000…100000]` meters.

Pitch, Roll, and Heading Offsets

Name: `/par/att/aoff`
Access: `rw`
Type: `{p,r,h}`
Default: `{0,0,0}`

`p, r` – Pitch- and Roll- offsets. Float values in the range `[-90…90]` degrees.

`h` – Heading offset. Float values in the range `[-180…180]` degrees.

Master Input Mode

Name: `/par/att/remote/imode`
Access: `rw`
Values: `none,rtcm,cmr`
Default: `none`

This parameter contains the input mode to use on the Master side.

Attitude Processing Mode

Name: `/par/att/procmode`
Access: `rw`
Type: `enumerated`
Values: `delay,extrap`
Default: `delay`

`delay` – there will be delays between time tags of the position and the attitude.

`extrap` – the time tags of the position and the attitude are aligned by using the extrapolation filter. Additional smoothing is added as a side effect.

Gain of the Attitude Extrapolation Filter

Name: `/par/att/gain`
Access: `rw`
Type: `float`
Values: `[0.1…10000]`
Default: `1`

Correlation Time of the Attitude Extrapolation Filter

Name: `/par/att/tcor`
Access: `rw`
Type: `float`
Values: `[0.001…10000]`
Default: `1`

Use Base Line Vectors with Fixed Ambiguities Only

Name: /par/att/fixed
Access: rw
Type: boolean
Values: on,off
Default: on

on – the attitude will be calculated with all three base line vectors being fixed only.

off – the attitude will be calculated even with some of base line vectors having float status.

Lever Arm Calculation Mode

Name: /par/att/armcalc
Access: rw
Type: enumerated
Values: on, off, auto
Default: off

on – the "lever arm" position, shifted from the antenna position by the lever arm vector, will be calculated and reported in the messages [RO], [RG], and [NR], provided the attitude is estimated.

off – if the attitude is not available, the NaN (Not a Number) values are reported. Also, NaN values are reported in the messages [RO], [RG], [NR].

auto – the messages [RO], [RG], [NR] report the position corrected by the rotated lever arm if the attitude is available, otherwise they report the position of the master antenna.

Lever Arm Vector

Name: /par/att/arm
Access: rw
Type: {n,r,d}
Default: {0,0,0}

The lever arm vector in the "nose", "right", "down" frame.

n, r, d – Nose-, Right-, Down- components of the lever arm vector. Float values in the range [-10000…10000] meters.

Ambiguity Fixing Statistics

The parameters described in this section are considered as technology parameters and are subject to change without notice. Note that they are primarily intended for test purposes.

This group of parameters provides the user with statistical (probabilistic) information on ambiguity fixing time. There are two different modes to obtain such information, specifically:

1. Enable *full firmware* reset once the ambiguities are fixed during the test cycle.
2. Enable *RTK only reset* once the ambiguities are fixed during the test cycle. This statistical information is presented as a histogram the i-th point of which specifies the probability of ambiguity fixing time being less than “i” seconds.

Precise Test Baseline

Name: `/par/pos/pd/hist/pr/C (C=[x|y|z])`
Type: `float [meters]`
Values: `[-1000000…1000000]`
Default: `0`

These three related parameters specify a priori precise coordinates of the test baseline in WGS-84 (X-, Y- and Z- coordinates, respectively).

The baseline vector’s components estimated in the current iteration are compared against the a priori precise coordinates. Ambiguities are considered fixed correctly if the difference between the a priori and a posteriori coordinates does not exceed R cm in each axis (note that R may be different in different firmware versions; it normally lies between 4 cm and 6 cm).

Ambiguity Fixing Statistics Using Full Firmware Resets

Name: `/par/pos/pd/hist/mode`
Access: `rw`
Values: `on,off`
Default: `off`

`on` – turn the mode on.

`off` – turn the mode off.

In this mode the receiver firmware is fully reset every time the integer ambiguities are re-fixed after the previous iteration. Although the actual time to ambiguity fix in the current iteration may well be less than one minute (and this time is correctly logged in the receiver memory), yet the new firmware reset is executed no sooner than in a minute after the previous one occurs. It is done in order for the receiver to be able to timely update ephemerides used. Should the current ambiguity fixing time exceed one minute, the firmware will fully reset immediately after the ambiguities are fixed.

This mode is available only when both `/par/raw/msint` and `/par/pos/msint` are set to 1000.

Erase the Current Statistics

Name: `/par/hist/reset`
Access: `w`
Values: `on`

Before you start collecting new statistics, you need to delete the previous ones. Setting this parameter to `on` will do this.

Estimated Probability of Ambiguities Fixing

Name: `/par/pos/pd/hist/out`
Access: `r`

Output data will look as follows:

```
⇐ 1<float> probability of fixing ambiguities in = 1 s
  2<float> probability of fixing ambiguities in = 2 s
  .
  .
  120<float> probability of fixing ambiguities in = 120 s
```

Total Number of Fixed Solutions

Name: `/par/pos/pd/hist/num`
Access: `r`

Output data will look as follows:

```
⇐ khist=<integer>
```

Percentage of Wrong Fixed Solutions

Name: `/par/pos/pd/hist/bad`
Access: `r`

Output data will look as follows:

```
⇐ bad=<float>
```

Note that if you have no a priori baseline coordinates then this estimate is of no avail.

Ambiguity Fixing Statistics Using RTK Engine Resets

Name: `/par/rtk/dbi/rest`
Access: `w`
Type: `integer`
Values: `0,1,2`
Default: `0`

This parameter is intended to enable the second ambiguity fixing time mode.

`0` – turn the mode off.

`1` – RTK engine will be reset on fixing the current set of ambiguities.

`2` – firmware will immediately output obtained statistics (histogram) to the current terminal and then start collecting data for the next histogram.

This parameter is not stored in NVRAM, therefore switching receiver off then on will turn this mode off.

Example: The following is an example histogram:

```
⇐ khist = 122 kbad = 0
  hist 1 0.000000
  hist 2 0.991803
  hist 3 0.991803
  hist 4 1.000000
  hist 5 1.000000
  …
  hist 120 1.000000
```

where:

`khist` – is the number of trials

`kbad` – is the number of the wrong fixes. Note that if you have no a priori baseline coordinates then this estimate is of no avail.

`hist` – 120 strings in the format “`hist n p`”, where `p` is the estimated probability of fixing ambiguity in no more than `n` seconds.

3.4.10 Reference Parameters

Parameters described in this section specify different kinds of reference information that could be used by multiple other receiver sub-systems. For example, reference coordinates could be sent from the reference station to rovers for RTK applications, are used to calculate RTCM corrections by DGPS reference station, and are used by the Improved Timing mode; reference antenna parameters could be utilized by both RTK base and RTK rover functionality; etc.

Reference Station Coordinates

Overview

Receiver supports separate reference station coordinates for GPS and GLONASS, as these two systems use different reference datums, WGS-84 and PE-90, respectively. In addition, reference station coordinates could be specified by the user int either Cartesian

or Geodetic system, that is also supported by means of separate parameters. Therefore, total of four customizable parameters are supported:

```
/par/ref/pos/gps/xyz
/par/ref/pos/gps/geo
/par/ref/pos/glo/xyz
/par/ref/pos/glo/geo
```

All these parameters should contain the coordinates of the L1 phase center of the receiver antenna.

While it’s possible to specify GPS and GLONASS reference positions independently, it is recommended to use one reference position estimate for both GPS and GLONASS in most cases. To simplify this common case, receiver supports simultaneous entry of both GPS and GLONASS reference coordinates through `/par/ref/pos//xyz` and `/par/ref/pos//geo` write-only parameters (note duplicated slashes in the parameter names).

Note that “xyz” and “geo” variants of the position for the same satellite system are mutually dependant. When one of the parameters is changed, another one is automatically re-calculated so that their values are always on the same datum and are consistent with each other.

While coordinates could be entered in any supported datum, receiver will need them in the datum used by particular satellite system, WGS-84 for GPS, and PE-90 for GLONASS, so it calculates those coordinates and makes them accessible for reading by the user in both Cartesian and Geodetic form through another four parameters:

```
/par/ref/syspos/gps/xyz
/par/ref/syspos/gps/geo
/par/ref/syspos/glo/xyz
/par/ref/syspos/glo/geo
```

With the parameters described hereafter the user specifies the location of the ARP. This location is then transmitted using RTCM 2.x message 24 and RTCM 3.x messages and is needed at the rover side in order to compute the RTK solution.

While the above parameters specify/describe the coordinates for L1 Antenna Phase Center (APC), the RTCM 2.x message 24 as well as RTCM 3.x standard requires that Antenna Reference Point (ARP) coordinates are to be transmitted from reference station to rover receivers. To meet this requirement, receiver supports additional set of parameters specifying ARP coordinates in the same way APC coordinates are specified. These parameters are:

```
/par/ref/arp/gps/xyz
/par/ref/arp/gps/geo
/par/ref/arp/glo/xyz
/par/ref/arp/glo/geo
```

and

```
/par/ref/sysarp/gps/xyz
/par/ref/sysarp/gps/geo
/par/ref/sysarp/glo/xyz
/par/ref/sysarp/glo/geo
```

The APC and ARP coordinates in the receiver are entirely independent. Receiver never calculates ARP coordinates from APC or vice versa. It's a duty of the user or corresponding application program to specify correct and consistent coordinates for ARP and APC.

Note that receiver doesn't use ARP coordinates except for the purpose of transmitting them in corresponding RTCM messages, while APC coordinates are essential to the receiver itself. Therefore, APC coordinates should always be entered for a reference station, while ARP coordinates may have arbitrary values unless you are going to transmit RTCM messages that carry ARP coordinates.

Parameters

Cartesian Reference Position for GPS

Name: `/par/ref/pos/gps/xyz`
Access: `rw`
Type: `pos_xyz`
Default: `{W84,+6378137.0000,+0.0000,+0.0000}`

Coordinates of L1 phase center of receiver antenna for GPS in Cartesian coordinate system.

Cartesian Reference Position for GLONASS

Name: `/par/ref/pos/glo/xyz`
Access: `rw`
Type: `pos_xyz`
Default: `{W84,+6378137.0000,+0.0000,+0.0000}`

Coordinates of L1 phase center of receiver antenna for GLONASS in Cartesian coordinate system.

Cartesian Reference Position for All Systems

Name: `/par/ref/pos//xyz`
Access: `w`
Type: `pos_xyz`

Setting this parameter will set both `/par/ref/pos/gps/xyz` and `/par/ref/pos/glo/xyz` to the specified value.

Geodetic Reference Position for GPS

Name: `/par/ref/pos/gps/geo`
Access: `rw`
Type: `pos_geo`
Default: `{W84,N00d00m00.000000s,E00d00m00.000000s,+0.0000}`

Coordinates of L1 phase center of receiver antenna for GPS in Geodetic coordinate system.

Geodetic Reference Position for GLONASS

Name: `/par/ref/pos/glo/geo`
Access: `rw`
Type: `pos_geo`
Default: `{W84,N00d00m00.000000s,E00d00m00.000000s,+0.0000}`

Coordinates of L1 phase center of receiver antenna for GLONASS in Geodetic coordinate system.

Geodetic Reference Position for All Systems

Name: `/par/ref/pos//geo`
Access: `w`
Type: `pos_geo`

Setting this parameter will set both `/par/ref/pos/gps/geo` and `/par/ref/pos/glo/geo` to the specified value.

WGS-84 Cartesian Reference Position (for GPS)

Name: `/par/ref/syspos/gps/xyz`
Access: `r`
Type: `pos_xyz`
Default: `{W84,+6378137.0000,+0.0000,+0.0000}`

WGS-84 Geodetic Reference Position (for GPS)

Name: `/par/ref/syspos/gps/geo`
Access: `r`
Type: `pos_geo`
Default: `{W84,N00d00m00.000000s,E00d00m00.000000s,+0.0000}`

PE-90 Cartesian Reference Position (for GLONASS)

Name: `/par/ref/syspos/glo/xyz`
Access: `r`
Type: `pos_xyz`
Default: `{P90,+6378137.0000,-6.3781,-1.0000}`

PE-90 Geodetic Reference Position (for GLONASS)

Name: `/par/ref/syspos/glo/geo`
Access: `r`
Type: `pos_geo`
Default: `{P90,S00d00m00.032557s,W000d00m00.206265s,+1.0000}`

Maximum Allowed Error in Reference Position

Name: `/par/ref/limit`
Access: `rw`
Type: `float [meters]`
Values: `[1…10000]`
Default: `1000`

Should the length of the vector connecting the current position calculated by receiver with that specified for the APC by the user exceed the maximum discrepancy level specified by this parameter, the reference station will stop transmitting any RTK or DGPS messages that depend on the quality of reference position.

Cartesian ARP Position for GPS

Name: `/par/ref/arp/gps/xyz`
Access: `rw`
Type: `pos_xyz`
Default: `{W84,+6378137.0000,+0.0000,+0.0000}`

Coordinates of ARP of receiver antenna for GPS in Cartesian coordinate system.

Cartesian ARP Position for GLONASS

Name: `/par/ref/arp/glo/xyz`
Access: `rw`
Type: `pos_xyz`
Default: `{W84,+6378137.0000,+0.0000,+0.0000}`

Coordinates of ARP of receiver antenna for GLONASS in Cartesian coordinate system.

Cartesian ARP Position for All Systems

Name: `/par/ref/arp//xyz`
Access: `w`
Type: `pos_xyz`

Setting this parameter will set both `/par/ref/arp/gps/xyz` and `/par/ref/arp/glo/xyz` to the specified value.

Geodetic ARP Position for GPS

Name: `/par/ref/arp/gps/geo`
Access: `rw`
Type: `pos_geo`
Default: `{W84,N00d00m00.000000s,E00d00m00.000000s,+0.0000}`

Coordinates of ARP of receiver antenna for GPS in Geodetic coordinate system.

Geodetic ARP Position for GLONASS

Name: `/par/ref/arp/glo/geo`
Access: `rw`
Type: `pos_geo`
Default: `{W84,N00d00m00.000000s,E00d00m00.000000s,+0.0000}`

Coordinates of ARP of receiver antenna for GLONASS in Geodetic coordinate system.

Geodetic ARP Position for All Systems

Name: `/par/ref/arp//geo`
Access: `w`
Type: `pos_geo`

Setting this parameter will set both `/par/ref/arp/gps/geo` and `/par/ref/arp/glo/geo` to the specified value.

WGS-84 Cartesian ARP Position (for GPS)

Name: `/par/ref/sysarp/gps/xyz`
Access: `r`
Type: `pos_xyz`
Default: `{W84,+6378137.0000,+0.0000,+0.0000}`

WGS-84 Geodetic ARP Position (for GPS)

Name: `/par/ref/sysarp/gps/geo`
Access: `r`
Type: `pos_geo`
Default: `{W84,N00d00m00.000000s,E00d00m00.000000s,+0.0000}`

PE-90 Cartesian ARP Position (for GLONASS)

Name: `/par/ref/sysarp/glo/xyz`
Access: `r`
Type: `pos_xyz`
Default: `{P90,+6378137.0000,-6.3781,-1.0000}`

PE-90 Geodetic ARP Position (for GLONASS)

Name: `/par/ref/sysarp/glo/geo`
Access: `r`
Type: `pos_geo`
Default: `{P90,S00d00m00.032557s,W000d00m00.206265s,+1.0000}`

Reference Position Averaging

All the features depending on the receiver reference position work best when specified reference position is known in advance with high precision. However, when precise position is unknown, there are still some applications that will tolerate even not that precise reference position. Reference Position Averaging feature allows receiver to automatically calculate and set its reference position.

Reference Position Averaging Mode

Name: `/par/ref/avg/mode`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `off`

`on` – receiver will compute its smoothed coordinates by averaging stand-alone position estimates over the specified interval after receiver reset or restart. The interval is defined by the `/par/ref/avg/span` parameter (see below). The absolute coordi-

nates thus estimated will then be automatically used as the receiver's reference position.

`off` – averaging mode is turned off.

Reference Position Averaging Interval

Name: `/par/ref/avg/span`
Access: `rw`
Type: `integer [seconds]`
Values: `[0…32000]`
Default: `180`

Provided `/par/ref/avg/mode` parameter is `on`, this parameter specifies time interval over which single-point position calculated by the receiver will be averaged before the result of averaging will be used as receiver reference position.

Reference Antenna Parameters

Marker to Antenna Phase Center (APC) Offset

Name: `/par/ref/ant/offs`
Access: `rw`
Type: `list {east,north,height}`

This parameter specifies the vector components between a surveyed point (land mark) and the APC.

`east` – east offset

`north` – north offset

`height` – height offset

East Offset of APC

Name: `/par/ref/ant/offs/east`
Access: `rw`
Type: `float [meters]`
Values: `[-100…100]`
Default: `0`

North Offset of APC

Name: `/par/ref/ant/offs/north`
Access: `rw`
Type: `float [meters]`
Values: `[-100…100]`
Default: `0`

Height Offset of APC

Name: `/par/ref/ant/offs/height`
Access: `rw`
Type: `float [meters]`
Values: `[-100…100]`
Default: `0`

Marker to the Antenna Reference Point (ARP) Offset

Name: `/par/ref/ant/arpoffs`
Access: `rw`
Type: `list {east,north,height}`

This parameter specifies the vector components between a surveyed point (land mark) and the ARP.

`east` – east offset

`north` – north offset

`height` – height offset

East Offset of ARP

Name: `/par/ref/ant/arpoffs/east`
Access: `rw`
Type: `float [meters]`
Values: `[-100.0…100.0]`
Default: `0`

North Offset of ARP

Name: `/par/ref/ant/arpoffs/north`
Access: `rw`
Type: `float [meters]`
Values: `[-100.0…100.0]`
Default: `0`

Height Offset of ARP

Name: `/par/ref/ant/arpoffs/height`
Access: `rw`
Type: `float [meters]`
Values: `[-100.0…100.0]`
Default: `0`

L1 APC to L2 APC Offset

Name: `/par/ref/ant/l2_l1`
Access: `rw`
Type: `list {east,north,height}`

This parameter specifies the vector components between L1 Antenna Phase Center (APC) and L2 APC.

`east` - east offset

`north` - north offset

`height` - height offset

East Offset of L2 APC

Name: `/par/ref/ant/l2_l1/east`
Access: `rw`
Type: `float [meters]`
Values: `[-0.1…0.1]`
Default: `0`

North Offset of L2 APC

Name: `/par/ref/ant/l2_l1/north`
Access: `rw`
Type: `float [meters]`
Values: `[-0.1…0.1]`
Default: `0`

Height Offset of L2 APC

Name: `/par/ref/ant/l2_l1/height`
Access: `rw`
Type: `float [meters]`
Values: `[-0.1…0.1]`
Default: `0`

Antenna type descriptor for RTCM 2.x and 3.0

Name: `/par/ref/ant/id`
Access: `rw`
Type: `string [0…31]`
Values: `up to 31 alphanumeric characters`
Default: `(empty string)`

Antenna Setup ID

Name:	`/par/ref/ant/setup`
Access:	`rw`
Type:	`integer`
Values:	`[0…255]`
Default:	`0`

This parameter is typically used by the differential service provider to inform the user about any change at the reference station that affects the antenna phase center variations.

Antenna Serial Number

Name:	`/par/ref/ant/sernum`
Access:	`rw`
Type:	`string [0…31]`
Values:	`up to 31 alphanumeric characters`
Default:	`(empty string)`

With this parameter the user specifies the individual antenna serial number.

3.4.11 Reference Station Data on Rover

Parameters described in this section represent information about reference station being used on rover receiver. They are mostly useful for RTK operation and serve two main purposes:

1. Allow the user of the rover receiver to get information that is received from the reference station.
2. Allow the user to enter information about reference station on rover and force the rover to use entered information instead of those received from reference station.

Many of these parameters could be considered to be a reflection on the rover side of the parameters described in “Reference Parameters” on page 143.

In this section, the data received from reference station is called *got* data. The data about reference station entered by the user on the rover is called *fixed* data. Receiver will select which data to actually use for RTK according to the values specified by the user for parameters that are described in the “Source of Data For Reference Station on Rover” below. The result of selection procedure is available through the parameters described in the “Reference Station Data for RTK” at the end of this section.

Data Received (Got) From Reference Station

Validity of Got Reference Position for GPS

Name: `/par/rover/base/pos/got/gps/valid`
Access: `r`
Type: `boolean`
Values: `on,off`
Default: `off`

`on` – rover has got valid GPS reference position from reference station.

`off` – rover didn't receive GPS reference position from reference station.

Got Reference Position (Cartesian) for GPS

Name: `/par/rover/base/pos/got/gps/xyz`
Access: `r`
Type: `pos_xyz`
Default: `{UNDEF,+6378137.0000,+0.0000,+0.0000}`

This parameter contains the base station's GPS reference coordinates received from the base station.

Got Reference Position (Geodetic) for GPS

Name: `/par/rover/base/pos/got/gps/geo`
Access: `r`
Type: `pos_geo`
Default: `{UNDEF,N00d00m00.000000s, E00d00m00.000000s,+0.0000}`

This parameter contains the base station's GPS reference coordinates received from the base station.

Validity of Got Reference Position for GLONASS

Name: `/par/rover/base/pos/got/glo/valid`
Access: `r`
Type: `boolean`
Values: `on,off`
Default: `off`

`on` – rover has got valid GLONASS reference position from reference station.

`off` – rover didn't receive GLONASS reference position from reference station.

Got Reference Position (Cartesian) for GLONASS

Name: `/par/rover/base/pos/got/glo/xyz`
Access: `r`
Type: `pos_xyz`
Default: `{UNDEF,+6378137.0000,+0.0000,+0.0000}`

This parameter contains the base station's GLONASS reference coordinates received from the base station.

The datum name `UNDEF` indicates that the truth reference coordinates are undefined or unavailable.

Got Reference Position (Geodetic) for GLONASS

Name: `/par/rover/base/pos/got/glo/geo`
Access: `r`
Type: `pos_geo`
Default: `{UNDEF,N00d00m00.000000s, E00d00m00.000000s,+0.0000}`

This parameter contains the base station's GLONASS reference coordinates received from the base station.

The datum name `UNDEF` indicates that the truth reference coordinates are undefined or unavailable.

Got Antenna ID

Name: `/par/rover/base/ant/got/id`
Access: `r`
Type: `string [0…31]`
Values: `(alphanumeric characters)`
Default: `(empty string)`

This parameter contains antenna ID received from reference station. If the antenna ID begins with a non-digit character, it is an RTCM antenna descriptor. Otherwise it is a CMR numerical antenna ID formatted as decimal.

Got Antenna Serial Number

Name: `/par/rover/base/ant/got/sernum`
Access: `r`
Type: `string [0…31]`
Default: `(empty string)`

This parameter contains antenna serial number received from reference station. The serial number could be transmitted in RTCM 2.x and RTCM 3.x.

Got Antenna Setup ID

Name: `/par/rover/base/ant/got/setup`
Access: `r`
Type: `integer`
Values: `[0…255]`
Default: `0`

This parameter contains antenna setup ID received from reference station. The setup ID could be transmitted in RTCM 2.x message 23 and RTCM 3.x.

Got Antenna Offset

Name: `/par/rover/base/ant/got/m_offs`
Access: `r`
Type: `list {type,val}`

This parameter contains reference antenna offset type and value received from reference station.

Got Antenna Offset Type

Name: `/par/rover/base/ant/got/m_offs/type`
Access: `r`
Type: `enumerated`
Values: `l1pc,arp`
Default: `l1pc`

This parameter contains reference antenna offset type.

`l1pc` – offset of antenna L1 phase center (APC)

`arp` – offset of antenna reference point (ARP)

Got Antenna Offset Value

Name: `/par/rover/base/ant/got/m_offs/val`
Access: `r`
Type: `list {east,north,height}`

This parameter contains antenna vector offset from the land mark to APC or ARP depending on the offset type used at the reference station.

`east` – east offset

`north` – north offset

`height` – height offset

Got East Antenna Offset Value

Name: `/par/rover/base/ant/got/m_offs/val/east`
Access: `rw`
Type: `float [meters]`
Values: `[-100…100]`
Default: `0`

Got North Antenna Offset Value

Name: `/par/rover/base/ant/got/m_offs/val/north`
Access: `rw`
Type: `float [meters]`
Values: `[-100…100]`
Default: `0`

Got Height Antenna Offset Value

Name: `/par/rover/base/ant/got/m_offs/val/height`
Access: `rw`
Type: `float [meters]`
Values: `[-100…100]`
Default: `0`

Got L1 APC to L2 APC Offset

Name: `/par/rover/base/ant/got/l2_l1`
Access: `r`
Type: `list {east,north,height}`

`east` – east offset

`north` – north offset

`height` – height offset

Got East Offset of L2 APC

Name: `/par/rover/base/ant/got/l2_l1/east`
Access: `rw`
Type: `float [meters]`
Values: `[-0.1…0.1]`
Default: `0`

Got North Offset of L2 APC

Name: `/par/rover/base/ant/got/l2_l1/north`
Access: `rw`
Type: `float [meters]`
Values: `[-0.1…0.1]`
Default: `0`

Got Height Offset of L2 APC

Name: `/par/rover/base/ant/got/l2_l1/height`
Access: `rw`
Type: `float [meters]`
Values: `[-0.1…0.1]`
Default: `0`

Data Entered (Fixed) For Reference Station

Fixed Cartesian Reference Position

Name: `/par/rover/base/pos/fix/xyz`
Access: `rw`
Type: `pos_xyz`
Default: `{W84,+6378137.0000,+0.0000,+0.0000}`

Fixed Geodetic Reference Position

Name: `/par/rover/base/pos/fix/geo`
Access: `rw`
Type: `pos_geo`
Default: `{W84,N00d00m00.000000s,`
`E00d00m00.000000s,+0.0000}`

Note: Currently only two datums, WGS-84 and PE-90, can be specified by means of this parameter.

Fixed Marker to Antenna Phase Center (APC) Offset

Name: `/par/rover/base/ant/fix/offs`
Access: `rw`
Type: `list {east,north,height}`
Default: `{0,0,0}`

`east` - east offset

`north` - north offset

`height` - height offset

Fixed East Antenna Offset

Name: `/par/rover/base/ant/fix/offs/east`
Access: `rw`
Type: `float [meters]`
Values: `[-100…100]`
Default: `0`

Fixed North Antenna Offset

Name: `/par/rover/base/ant/fix/offs/north`
Access: `rw`
Type: `float [meters]`
Values: `[-100…100]`
Default: `0`

Fixed Height Antenna Offset

Name: `/par/rover/base/ant/fix/offs/height`
Access: `rw`
Type: `float [meters]`
Values: `[-100…100]`
Default: `0`

Fixed L1 APC to L2 APC Offset

Name: `/par/rover/base/ant/fix/l2_l1`
Access: `r`
Type: `list {east,north,height}`

`east` – east offset

`north` – north offset

`height` – height offset

Fixed East Offset of L2 APC

Name: `/par/rover/base/ant/fix/l2_l1/east`
Access: `rw`
Type: `float [meters]`
Values: `[-0.1…0.1]`
Default: `0`

Fixed North Offset of L2 APC

Name: `/par/rover/base/ant/fix/l2_l1/north`
Access: `rw`
Type: `float [meters]`
Values: `[-0.1…0.1]`
Default: `0`

Fixed Height Offset of L2 APC

Name: `/par/rover/base/ant/fix/l2_l1/height`
Access: `rw`
Type: `float [meters]`
Values: `[-0.1…0.1]`
Default: `0`

Source of Data For Reference Station on Rover

Clear the Reference Station Coordinates

Name: `/par/pos/pd/ref/clean`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `on`

This parameter is used to clear the currently effective coordinates of the reference station.

`on` – RTK engine will assume there are no reference coordinates and therefore will disable differential positioning until next reference coordinates are received or entered by the user. The value of the parameter is immediately reset to `off`.

`off` – ignored.

Reference Position Source

Name: `/par/rover/base/pos/cur`
Access: `rw`
Type: `enumerated`
Values: `got,fix`
Default: `got`

`got` – reference position received from reference station will be used.

`fix` – reference position entered for reference station by the user will be used.

Reference Position Set Source

Name: `/par/ref/src`
Access: `rw`
Values: `gps,glo,any`
Default: `any`

This parameter instructs the rover which of the reference position sets received from the base to select for use by RTK.

`gps` – GPS reference position will be selected for use by RTK, GLONASS reference position will be ignored.

`glo` – GLONASS reference position will be selected for use by RTK, GPS reference position will be ignored.

`any` – any reference position received from base station will be selected for use by RTK.

Antenna Offset Source

Name: `/par/rover/base/ant/cur`
Access: `rw`
Type: `enumerated`
Values: `got,fix`
Default: `got`

`got` – antenna offsets received from reference station will be used.

`fix` – antenna offsets entered by the user will be used.

Both L1 APC to L2 APC offset and Marker to APC offset usage are affected by this parameter.

Reference Station Data for RTK

Reference Position (Cartesian) for RTK

Name: `/par/pos/pd/ref/pos/xyz`
Access: `r`
Type: `pos_xyz`
Default: `{UNDEF,+6378137.0000,+0.0000,+0.0000}`

This parameter contains the base station's reference position (in Cartesian form) to be used by RTK.

The datum name `UNDEF` indicates that the truth reference coordinates are undefined or unavailable.

Reference Position (Geodetic) for RTK

Name: `/par/pos/pd/ref/pos/geo`
Access: `r`
Type: `pos_geo`
Default: `{UNDEF,N00d00m00.000000s, E00d00m00.000000s,+0.0000}`

This parameter contains the base station's reference position (in Geodetic form) to be used by RTK.

The datum name `UNDEF` indicates that the truth reference coordinates are undefined or unavailable.

Antenna Offset for RTK

Name: `/par/pos/pd/ref/ant/m_offs`
Access: `r`
Type: `list {type,val}`

This parameter contains reference antenna offset type and value to be used for RTK.

Antenna Offset Type for RTK

Name: `/par/pos/pd/ref/ant/m_offs/type`
Access: `r`
Type: `enumerated`
Values: `l1pc,arp`
Default: `l1pc`

This parameter contains reference antenna offset type to be used for RTK.

`l1pc` – offset of antenna L1 phase center (APC)

`arp` – offset of antenna reference point (ARP)

Antenna Offset Value for RTK

Name: `/par/pos/pd/ref/ant/m_offs/val`
Access: `r`
Type: `list {east,north,height}`

This parameter contains antenna vector offset from the land mark to APC or ARP depending on the offset type to be used for RTK.

`east` – east offset

`north` – north offset

`height` – height offset

East Antenna Offset Value for RTK

Name: `/par/pos/pd/ref/ant/m_offs/val/east`
Access: `rw`
Type: `float [meters]`
Values: `[-100…100]`
Default: `0`

North Antenna Offset Value for RTK

Name: `/par/pos/pd/ref/ant/m_offs/val/north`
Access: `rw`
Type: `float [meters]`
Values: `[-100…100]`
Default: `0`

Height Antenna Offset Value for RTK

Name: `/par/pos/pd/ref/ant/m_offs/val/height`
Access: `rw`
Type: `float [meters]`
Values: `[-100…100]`
Default: `0`

L1 APC to L2 APC Offset for RTK

Name: `/par/pos/pd/ref/ant/l2_l1`
Access: `r`
Type: `list {east,north,height}`

`east` – east offset

`north` – north offset

`height` – height offset

East Offset of L2 APC for RTK

Name: `/par/pos/pd/ref/ant/l2_l1/east`
Access: `rw`
Type: `float [meters]`
Values: `[-0.1…0.1]`
Default: `0`

North Offset of L2 APC for RTK

Name: /par/pos/pd/ref/ant/l2_l1/north
Access: rw
Type: float [meters]
Values: [-0.1…0.1]
Default: 0

Height Offset of L2 APC for RTK

Name: /par/pos/pd/ref/ant/l2_l1/height
Access: rw
Type: float [meters]
Values: [-0.1…0.1]
Default: 0

3.4.12 Antenna Database

JAVAD GNSS receivers contain embedded antenna data-base. This database provides identification and measurement information for more than 200 antennas. More precisely, for each antenna in this database, the following entries are given:

- Antenna identifier used in RTCM standard version 3.0. This identifier or ID (as denoted below) is a string comprising up to 20 characters.
- Antenna identifier used in CMR standard. This identifier or CMR ID (as denoted below) is an integer value in the range of 0…255.
- Vector offset between the ARP and L1 phase center
- Vector offset between L1 and L2 phase centers

Note: The following parameters are supported by all receiver boards except for HE_GG and HE_GD, JNS100, JNS100GG.

Antenna Type Definitions

Name: /par/antdb
Access: r
Type: list {ver,id,cmr,ids}

Antenna Database Version

Name: /par/antdb/ver
Access: r
Type: string

The antenna database version in the M.N.K format, where:

`M` – database major version as decimal

`N` – database minor version as decimal

`K` – patch level as decimal

Antenna Database Entries

Name: `/par/antdb`
Access: `r`
Type: `list {id,cmr,ids}`

`id` – parameters for antennas by ID

`cmr` – parameters for antennas by CMR ID

`ids` – maps of ID to CMR ID, and CMR ID to ID

Parameters for Antennas by ID

Name: `/par/antdb/id`
Access: `r`
Type: `list {[ID]}`

The list of all the antenna IDs included in the database along with their parameters.

Parameters for Antenna ID

Name: `/par/antdb/id/[ID]`
Access: `r`
Type: `list {cmr,l1_arp,l2_l1}`

`cmr` – corresponding CMR antenna identifier.

`l1_arp` – vector offset between L1 phase center and antenna reference point.

`l2_l1` – vector offset between L2 and L1 phase centers.

Antenna CMR ID

Name: `/par/antdb/id/[ID]/cmr`
Access: `r`
Type: `integer`
Values: `[0…255] or empty string`

The CMR antenna identifier for specified antenna ID. If the selected antenna ID does not have the corresponding CMR identifier, this parameter is set to an empty string.

L1 to ARP Offset

Name: `/par/antdb/id/[ID]/l1_arp`
Access: `r`
Type: `list {east, north, height} of float [meters]`
Values: `{[-100.0…100.0],[-100.0…100.0],[-100.0…100.0]}`

Vector offset between L1 phase center and ARP.

L1 to L2 Offset

Name: `/par/antdb/id/[ID]/l2_l1`
Access: `r`
Type: `list {east, north, height} of float [meters]`
Values: `{[-100.0…100.0], [-100.0…100.0],[-100.0…100.0]}`

Vector offset between L1 and L2 phase centers.

Parameters for Antennas by CMR ID

Name: `/par/antdb/cmr`
Access: `r`
Type: `list {[CMR_ID]}`

The list of all antennas with assigned CMR ID along with their parameters.

Parameters for Antenna CMR ID

Name: `/par/antdb/cmr/[CMR_ID]`
Access: `r`
Type: `list {id,l1_arp,l2_l1}`

`id` – corresponding antenna ID.

`l1_arp` – vector offset between L1 phase center and antenna reference point.

`l2_l1` – vector offset between L2 and L1 phase centers.

Antenna ID for Specific CMR ID

Name: `/par/antdb/cmr/[CMR_ID]/id`
Access: `r`
Type: `string`

Antenna ID for the specified CMR ID.

L1 to ARP Offset for CMR ID.

Name: `/par/antdb/cmr/[CMR_ID]/l1_arp`
Access: `r`
Type: `list {east, north, height} of float [meters]`
Values: `{[-100.0…100.0],[-100.0…100.0],[-100.0…100.0]}`

Vector offset between L1 phase center and ARP.

L1 to L2 Offset for CMR ID

Name: /par/antdb/cmr/[CMR_ID]/l2_l1
Access: r
Type: list {east, north, height} of float [meters]
Values: {[-100.0…100.0], [-100.0…100.0],[-100.0…100.0]}

Vector offset between L1 and L2 phase centers.

IDs Maps

Name: /par/antdb/ids
Access: r
Type: list {id,cmr}

id – map from antenna ID to CMR antenna ID

cmr – map from CMR antenna ID to antenna ID

ID to CMR ID Map

Name: /par/antdb/ids/id
Access: r
Type: list {[ID]}

For every antenna ID contains its corresponding antenna ID and CMR_ID.

ID to CMR ID Map Element

Name: /par/antdb/ids/id/[ID]
Access: r
Type: string

For specified antenna ID, contains antenna ID together with the corresponding CMR ID in the format {ID,CMR_ID}.

CMR ID to ID Map

Name: /par/antdb/ids/cmr
Access: r
Type: list{[CMR_ID]}

For every antenna with known CMR ID contains its corresponding antenna ID and CMR_ID.

CMR ID to ID Map Element

Name: /par/antdb/ids/cmr/[CMR_ID]
Access: r
Type: string

For specified CMR antenna ID, contains antenna ID together with the corresponding CMR ID in the format `{ID,CMR_ID}`.

3.4.13 Base and Rover Modes

Traditionally, many GNSS receivers support the notion of *base mode* and *rover mode*. For example, when receiver is used as RTCM DGPS reference station, it's not unusual to say that receiver works in *RTCM DGPS base mode*, and when receiver is computing DGPS position, it works in so called *RTCM DGPS rover mode*. However, such model is too simplistic to be used to exactly specify the required behavior of JAVAD GNSS receivers. For example, using this terminology, a JAVAD GNSS receiver is capable to work as, say, CMR RTK base, DGPS RTCM base, and RTCM3 RTK rover simultaneously.

Due to their flexibility, JAVAD GNSS receivers have no notion of base or rover modes internally. We can say that they are *modeless* in this sense. On one hand, this allows applications to decide exact meaning of base and rover modes themselves, if they wish to. On the other hand, this makes it somewhat more difficult to design such applications. To simplify development of applications utilizing notion of base/rover modes, JAVAD GNSS receivers support a set of parameters that:

1. Allow to check if some feature that could be considered to belong to either base or rover mode is active.
2. Allow to turn *off* some features that could be considered to belong to either base or rover mode.

Note that these parameters can't be used to turn base or rover mode *on*, because JAVAD GNSS receivers have no idea what exactly those modes are from the point of view of given application.

Example: Turn off base and rover mode:

⇒ `set,/par/base/mode/,off`

⇒ `set,/par/rover/mode/,off`

⇒ `set,/par/pos/mode/cur,sp`

●

Base Modes

Name:	`/par/base/mode`
Access:	`rw`
Type:	`list {rtcm,cmr,jps,rtcm3}`
Values:	`{on|off,on|off,on|off,on|off}`
Default:	`{off,off,off,off}`

This parameter is a list of boolean values describing the status of output of messages of corresponding formats. You can turn off all of these formats by using the command `set,/par/base/mode/,off`. Refer to description of individual parameters below for details.

RTCM 2.x Base Mode

Name: `/par/base/mode/rtcm`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `off`

Setting this parameter to `off` will disable output of all the RTCM 2.x messages to all of the receiver ports. Receiver returns error if you try to set this parameter to `on`.

`on` – indicates that there is at least one RTCM 2.x message enabled to be output to at least one of receiver ports.

`off` – indicates that there are no RTCM 2.x messages enabled for output.

CMR Base Mode

Name: `/par/base/mode/cmr`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `off`

Setting this parameter to `off` will disable output of all the CMR messages to all of the receiver ports. Receiver returns error if you try to set this parameter to `on`.

`on` – indicates that there is at least one CMR message enabled to be output to at least one of receiver ports.

`off` – indicates that there are no CMR messages enabled for output.

GREIS Base Mode

Name: `/par/base/mode/jps`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `off`

Setting this parameter to `off` will disable output of all the GREIS messages to all of the receiver ports to which GREIS [BI] is enabled. Receiver returns error if you try to set this parameter to `on`.

`on` – indicates that GREIS [BI] message is enabled to be output to at least one of receiver ports.

`off` – indicates that GREIS [BI] message is not enabled for output.

RTCM 3.x Base Mode

Name: `/par/base/mode/rtcm3`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `off`

Setting this parameter to `off` will disable output of all the RTCM 3.x messages to all of the receiver ports. Receiver returns error if you try to set this parameter to `on`.

`on` – indicates that there is at least one RTCM 3.x message enabled to be output to at least one of receiver ports.

`off` – indicates that there are no RTCM 3.x messages enabled for output.

Rover Mode

Name: `/par/rover/mode`
Access: `rw`
Type: `list {rtcm,cmr,jps,rtcm3}`
Values: `{on|off,on|off,on|off,on|off}`
Default: `{off,off,off,off}`

This parameter is a list of boolean values describing the status of input modes of receiver ports. You can turn all the receiver ports that are currently in `rtcm`, `cmr`, `jps`, or `rtcm3` input modes to `cmd` mode by using the command `set,/par/rover/mode/,off`. Refer to description of individual parameters below for details.

RTCM 2.x Rover Mode

Name: `/par/rover/mode/rtcm`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `off`

Setting this parameter to `off` will switch all of the ports running in `rtcm` input mode back to `cmd` mode. Receiver returns error if you try to set this parameter to `on`.

`on` – indicates that at least one receiver port is set to `rtcm` input mode.

`off` – indicates that none of the receiver ports are set to `rtcm` input mode.

CMR Rover Mode

Name: `/par/rover/mode/cmr`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `off`

Setting this parameter to `off` will switch all of the ports running in `cmr` input mode back to `cmd` mode. Receiver returns error if you try to set this parameter to `on`.

`on` – indicates that at least one receiver port is set to `cmr` input mode.

`off` – indicates that none of the receiver ports are set to `cmr` input mode.

GREIS Rover Mode

Name: `/par/rover/mode/jps`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `off`

Setting this parameter to `off` will switch all of the ports running in `jps` input mode back to `cmd` mode. Receiver returns error if you try to set this parameter to `on`.

`on` – indicates that at least one receiver port is set to `jps` input mode.

`off` – indicates that none of the receiver ports are set to `jps` input mode.

RTCM 3.x Rover Mode

Name: `/par/rover/mode/rtcm3`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `off`

Setting this parameter to `off` will switch all of the ports running in `rtcm3` input mode back to `cmd` mode. Receiver returns error if you try to set this parameter to `on`.

`on` – indicates that at least one receiver port is set to `rtcm3` input mode.

`off` – indicates that none of the receiver ports are set to `rtcm3` input mode.

3.4.14 RTCM 2.x Parameters

RTCM 2.x Reference Station Parameters

RTCM 2.x Version to Use for RTCM 2.x Messages

Name: `/par/rtcm/base/ver`
Access: `rw`
Type: `enumerated`
Values: `v2.1,v2.2,v2.3`
Default: `v2.3`

This parameter allows you to use JAVAD GNSS receivers together with legacy third-party rover receivers that don't support higher versions of RTCM 2.x standard. Note that only RTK messages are affected.

Zero the Rate of Change of Pseudorange Corrections for GPS

Name: `/par/rtcm/base/zrate/gps`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `off`

`on` – receiver will set the rate of change of the pseudorange corrections to zero for GPS satellites in RTCM 2.x message types 1, 9, 31, and 34. In some cases it may improve DGPS accuracy.

`off` – receiver will put computed values into the messages.

Zero the Rate of Change of Pseudorange Corrections for GLONASS

Name: `/par/rtcm/base/zrate/glo`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `off`

`on` – receiver will set the rate of change of the pseudorange corrections to zero for GLONASS satellites in RTCM 2.x message types 1, 9, 31, and 34. In some cases it may improve DGPS accuracy.

`off` – receiver will put computed values into the messages.

Use Local Datum for Referencing Differential Corrections

Name: `/par/rtcm/base/locdtm`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `off`

`on` – the datum specified by the `/par/pos/datum/cur` parameter will be used for referencing GPS and GLONASS differential corrections.

`off` – receiver will use WGS-84 and PE-90 datums for referencing GPS and GLONASS corrections, respectively.

Note: Ensure that rover uses the same setting for its `/par/rtcm/rover/locdtm` parameter. If use of local datum is enabled, the same local datum should be specified at both the base station and the rover (see `/par/pos/datum/cur` parameter on page 91).

This parameter affects RTCM 2.x message types 1, 9, 20, 21, 31 and 34.

The RTCM 2.x standard recommends using WGS-84 and PE-90 for referencing GPS and GLONASS differential corrections, respectively[9]. In some cases, however, it can be desirable to transmit corrections referenced to a local datum.

For example, in code differential, coordinates of the base station can be given in a local datum. In this case, corrections referenced to this local datum may be transmitted to the rover. At the rover side, provided that the same local datum is chosen, a user can obtain the position expressed in the same local datum. Thus, it is possible to obtain the coordinates, expressed in a local datum, without any transformations from local datum to, say, WGS-84 datum prior to transmitting the corrections. Thus, this procedure provides a comfortable method for obtaining coordinates, expressed in a local datum. However, some limitations of this procedure should be mentioned:

1. The rover should “know” that differential corrections are referenced to a local datum. If a base station serves as a reference for many rovers, each of those rovers should use the local datum specified at the base station, otherwise the rover coordinates can be distorted.
2. On long baselines, referencing the differential corrections to a local datum may introduce an additional error in the coordinates.

9. See RTCM recommended standards for differential GNSS (Global Navigation Satellite System) service, version 2.3, August 20, 2001. (RTCM PAPER 136-2001/SC104-STD).

Satellite Constellation for RTCM 2.x Messages

Name:	`/par/rtcm/base/sys`
Access:	`rw`
Type:	`array [0…2] of boolean`
Values:	`{on|off,on|off}`
Default:	`{on,on}`

This parameter instructs the base receiver to include in RTCM 2.x message types 18, 19, 20 and 21 only data associated with the specified satellite constellation. The first and the second values correspond to GPS and GLONASS, respectively. By default, all of available GPS and GLONASS satellites will be taken into account when generating these message types.

Maximum Number of Satellites for RTCM 2.x Messages

Name:	`/par/rtcm/base/svm`
Access:	`rw`
Type:	`integer`
Values:	`[0…127]`
Default:	`0`

This parameter affects message types 18, 19, 20, and 21. It allows to save bandwidth of slow communication channels.

`0` – all of the available satellites will be included in corresponding messages.

`[1…127]` – not more than the specified number of satellites will be included in corresponding messages. The satellites that will be excluded are those with lowest elevations.

RTCM 2.x Base Station Health

Name:	`/par/rtcm/base/health`
Access:	`rw`
Type:	`enumerated`
Values:	`good,bad,unknown`
Default:	`good`

The values correspond to the following terms of the RTCM 2.x standard:

`good` – normal performance

`bad` – health status is “reference station not working”

`unknown` – health status is “reference station transmission not monitored”.

RTCM 2.x Base Station Identifier

Name: `/par/rtcm/base/stid`
Access: `rw`
Type: `integer`
Values: `[0…1023]`
Default: `0`

Text for RTCM 2.x Message Types 16 and 36 (GPS)

Name: `/par/rtcm/base/text/gps`
Access: `rw`
Type: `string[0…90]`
Default: `(empty string)`

Text for RTCM 2.x Message Types 16 and 36 (GLONASS)

Name: `/par/rtcm/base/text/glo`
Access: `rw`
Type: `string[0…90]`
Default: `(empty string)`

Enable CA/L1 in RTCM 2.x Message Types 18 through 21

Name: `/par/rtcm/base/meas/ca`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `on`

Enable P/L1 in RTCM 2.x Message Types 18 three 21

Name: `/par/rtcm/base/meas/p1`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `off`

Enable P/L2 in RTCM 2.x Message Types 18 three 21

Name: `/par/rtcm/base/meas/p2`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `on`

Use Smoothed Pseudoranges in RTCM 2.x Message Types 19 three 21

Name: `/par/rtcm/base/smooth`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `off`

Enable Delimiting Characters for RTCM 2.x Messages

Name: `/par/rtcm/base/end`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `off`

`on` – receiver will insert up to two delimiting characters at the end of every RTCM 2.x message (these characters are specified by the value of `/par/rtcm/base/es` parameter, see below).

Delimiting Character(s) for RTCM 2.x Messages

Name: `/par/rtcm/base/es`
Access: `rw`
Type: `list {integer,integer}`
Values: `{[-1…127],[-1…127]}`
Default: `{13,10}`

This parameter determines up to two delimiting characters that will be added to the end of every RTCM 2.x message. The value `-1` disables corresponding character, other values specify ASCII code of the character.

RTCM 2.x Rover Parameters

Use Local Datum for Referencing Differential Corrections on Rover

Name: `/par/rtcm/rover/locdtm`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `off`

The value of this parameter should match those of the `/par/rtcm/base/locdtm` parameter on the reference station. Refer to the description of the aforementioned parameter on page 173 for details.

Use Not Monitored Reference Station

Name: `/par/rtcm/rover/usenm`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `on`

This parameter enables/disables use of data received from a reference station whose health status code is set to 110. According to the RTCM 2.x standard, this code indicates that data transmitted by this station is “not monitored.”

`on` – use data from a reference station even when its status code is 110.

`off` – do not use data from a reference station which status code is 110.

Check Sequence Number From the RTCM 2.x Messages

Name: `/par/rtcm/rover/seqnum`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `on`

`on` – use message sequence numbers in calculation of data link quality.

`off` – do not use message sequence numbers in calculation of data link quality.

An RTCM 2.x message has a data field called sequence number. Sequence numbers allow the receiver to check whether any messages have been lost when receiving RTCM 2.x data. Such checking is enabled by default. If the receiver detects that a message is lost (i.e. the difference between the current and the previous sequence numbers is not equal to unity), the “bad message counter” will be incremented. The data link quality is available in GREIS [DL] message described on page 328.

RTCM 2.x Version to Assume at the Rover

Name: `/par/rtcm/rover/ver`
Access: `rw`
Type: `enumerated`
Values: `v2.1,v2.2,v2.3`
Default: `v2.3`

This parameter allows you to use JAVAD GNSS receivers together with reference stations that transmit messages in the format specified by older versions of RTCM 2.x standard. Note that only RTK messages are affected.

Multiple Message Indicator Mode

Name: `/par/rtcm/rover/mmi`
Access: `rw`
Type: `enumerated`
Values: `def,on,off`
Default: `off`

RTCM 2.x message types 18/19 and 20/21 have a flag called “Multiple Message Indicator”. This flag serves to identify the last message in a group of such messages referenced to the same time. Unfortunately, different manufacturers have interpreted this flag differently, which resulted in incompatibility between the formats used by different developers. The version 2.3 of the RTCM 2.x standard[10], unlike the version 2.2, defines this flag explicitly and unambiguously. This flag will allow a JAVAD GNSS receiver configured as a rover to be capable of using RTCM 2.3 messages transmitted by other manufacturers’ base receivers.

`def` – this is the same as `on` when RTCM version (as defined by the `/par/rtcm/rover/ver` parameter) is set to either v2.1 or v2.2, and is the same as `off` when RTCM version is set to v2.3.

`on` – receiver will always verify the flag. This is expected to reduce the latency time since in this case the rover receiver needn’t wait for arriving RTCM 2.x messages referenced to the next epoch. Note, however, that this will be possible only on condition that the Multiple Message Indicator behaves exactly as it is specified in version 2.3. Otherwise the data received may be interpreted incorrectly.

`off` – receiver will have to wait[11] for RTCM 2.x data corresponding to the next epoch to arrive to accept the data from the current epoch.

Note: The default value for this parameter allows rover to work reliably with any version of RTCM messages transmitted by the reference station. It is recommended to set the mode to `on` only if it is known that the base receiver outputs RTCM version 2.3. Otherwise selecting this mode may cause malfunction of the rover receiver in RTK.

Enable Complete Epoch Received Logic

Name: `/par/rtcm/rover/full`
Access: `rw`
Type: `enumerated`
Values: `def,on,off`
Default: `def`

This parameter specifies if the *complete epoch received* logic is applied while the receiver decoding RTCM data for RTK. The term *complete epoch received* means the

10. RTCM PAPER 136-2001/SC104-STD
11. Though this could be sometimes compensated by the *complete epoch received* logic described later.

receiver obtained all the data necessary for the given epoch. For example, for a GPS/GLONASS dual frequency receiver the term means the receiver has acquired the code and phase measurements on both frequencies for at least one satellite for either constellation.

`def` – the logic is either turned on or off depending on the parameter `/par/rtcm/rover/ver`, — it is turned off for the RTCM versions v2.1 and v2.2, and is turned on for version v2.3.

`on` – the logic is turned on.

`off` – the logic is turned off.

In RTCM versions earlier than 2.3, due to different interpretation of the RTCM 2.x standard by different manufacturers, it is not always possible to identify the end of epoch based on the RTCM format itself. In this case the receiver can either wait when an RTCM 2.x message with a different time arrives (thus increasing latency), or apply the *complete epoch received* logic. The latter can decrease the RTK corrections' latency by eliminating the delay required for receiving the first message referenced to the next epoch (typically 1 second).

Source of Antenna Reference Position

Name:	`/par/rtcm/rover/refsrc`
Access:	`rw`
Type:	`enumerated`
Values:	`auto,l1pc,arp`
Default:	`auto`

This parameter allows to maintain compatibility between various ways of expressing the reference antenna position.

Note: JAVAD GNSS recommends that you use this parameter's default value unless you are completely sure of the message set being transmitted by the reference station.

The RTCM standard version 2.3 supports new message type 24 which provides the exact location of the reference station and the antenna height as the distance to the Antenna Reference Point (ARP). Remember that the previous versions of the standard use the message types 3 and 32 to broadcast the coordinates of the reference antenna and these messages contain the coordinates of the Antenna L1 Phase Center (APC).

`auto` – if both message sets (3/31/22 and 23/24) are transmitted in the same data stream, the rover receiver will use ARP coordinates (message type 24). If only one of the message sets is transmitted, the rover receiver will automatically extract the antenna coordinates available in the given data stream and applies them to the RTK engine.

`l1pc` – receiver will use the coordinates of the APC extracted from message types 3 or 32.

`arp` – receiver will use the coordinates of the ARP extracted from message type 24.

Reset the RTCM 2.x Decoders

Name:	`/par/rtcm/rover/reset`
Access:	`rw`
Type:	`boolean`
Values:	`on,off`
Default:	`off`

`on` – receiver will reset the RTCM decoders and then will restore the value `off` of this parameter.

`off` – ignored.

For example, you may use this parameter to reset the logic associated with the value used in the `/par/rtcm/rover/refsrc` parameter.

3.4.15 RTCM 3.x Parameters

RTCM 3.x Reference Station Parameters

RTCM 3.x Reference Station Identifier

Name:	`/par/rtcm3/base/stid`
Access:	`rw`
Type:	`integer`
Values:	`[0…4095]`
Default:	`0`

This parameter contains the reference station ID that will be part of the RTCM 3.x correction messages. On the rover side, this ID allows easy identification of the reference station whose RTCM 3.x messages are being received by the rover.

Text for RTCM 3.x Message

Name:	`/par/rtcm3/base/text`
Access:	`rw`
Type:	`string[0…127]`
Values:	`arbitrary`
Default:	`(empty string)`

The value of this parameter will be included into proprietary RTCM 3.x text message (Message Type 4091).

RTCM 3.x Rover Parameters

There are currently none.

3.4.16 CMR Parameters

CMR Reference Station Parameters

Receiver Motion State for CMR

Name: `/par/cmr/base/motion`
Access: `rw`
Type: `enumerated`
Values: `unknown,static,kinematic`
Default: `unknown`

`unknown` – motion state is undefined. Corresponding CMR messages will contain reference coordinates entered by the user.

`static` – motion state is static (i.e., not moving). Corresponding CMR messages will contain reference coordinates entered by the user.

`kinematic` – motion state is moving. Corresponding CMR messages will use the current position estimate computed by the receiver (it can be an RTK, DGPS or single-point position estimate depending on which positioning mode is on) for reference station coordinates.

Data for CMR Message Type 2

Name: `/par/cmr/base/desc`
Access: `rw`
Type: `{string[0…8],string[0…16],string[0…50]}`
Values: `{(arbitrary),(arbitrary),(arbitrary)}`
Default: `{"","",""}`

This parameter contains three strings specifying “short station ID”, “COGO code”, and “long station ID”, in this order.

CMR Reference Station Identifier

Name: `/par/cmr/base/stid`
Access: `rw`
Type: `integer`
Values: `[0…31]`
Default: `0`

Maximum Number of Satellites for CMR Messages

Name: `/par/cmr/base/svm`
Access: `rw`
Type: `integer`
Values: `[0…127]`
Default: `0`

This parameter affects CMR messages containing per-satellite data. It allows to save bandwidth of slow communication channels.

`0` – all of the available satellites will be included in corresponding messages.

`[1…127]` – not more than the specified number of satellites will be included in corresponding messages. The satellites that will be excluded are those with lowest elevations.

Enable P/L2 in CMR Messages

Name: `/par/cmr/base/meas/p2`
Access: `rw`
Type: `boolean`
Values: `on, off`
Default: `on`

Substitute P/L1 for CA/L1 in CMR Messages

Name: `/par/cmr/base/pcode`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `off`

Type of CMR Message to Use for GLONASS

Name: `/par/cmr/base/glo/type`
Access: `rw`
Type: `integer`
Values: `[3…7]`
Default: `3`

Since the CMR format does not allow for any predefined message type for GLONASS measurements, you must specify a message type for GLONASS raw data on your own. This is the purpose that this parameter serves.

Since some new CMR message types may appear in the future, be sure that the message type assigned to GLONASS measurements is different from all the other CMR message types. Should a conflict due to ambiguous message types occur, you may need to re-

define the message type associated with GLONASS measurements (just choose any unused number within a range of [3…7]).

In addition, ensure that both the reference station and the rover receiver use the same message type for GLONASS data (see `/par/cmr/rover/glo/type` below).

CMR Antenna Type

Name: `/par/cmr/base/ant/type`
Access: `rw`
Type: `integer`
Values: `[0…255]`
Default: `0`

This parameter contains CMR antenna numeric identifier for the type of antenna being used at the reference station.

Note: To find out relationship between your antenna type and the corresponding CMR antenna ID, use, for example, the `/par/antdb/ids` parameter.

CMR Receiver Type

Name: `/par/cmr/base/rcv/type`
Access: `rw`
Type: `integer`
Values: `[0…255]`
Default: `0`

This parameter contains CMR receiver numeric identifier for the type of receiver being used at the reference station.

CMR Plus Reference Station Compatibility

Name: `/par/cmr/base/idxset`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `off`

This parameter is used to preserve backward compatibility with the JNS firmware version 2.2 when working with CMR Plus messages.

`on` – instructs receiver to code CMR Plus message in accordance with the firmware version 2.2, thus, rover receivers which are uploaded with the version 2.2 can work properly.

`off` – use standard format for CMR Plus message.

CMR Rover Parameters

Type of CMR Message to Expect for GLONASS

Name: `/par/cmr/rover/glo/type`
Access: `rw`
Type: `integer`
Values: `[3…7]`
Default: `3`

See `/par/cmr/base/glo/type` above for details.

CMR Plus Rover Compatibility

Name: `/par/cmr/rover/idxset`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `off`

This parameter is used to preserve compatibility with the JNS firmware version 2.2 when working with CMR Plus message.

`on` – receiver will decode CMR Plus message in accordance with the firmware version 2.2, thus, rover receivers can work with the reference receiver, which is uploaded with the firmware version 2.2 and newer.

`off` – receiver will decode CMR Plus messages in a standard way.

3.4.17 Parameters of Generic GREIS Messages

Masks and Counters

Elevation Mask for Measurements Output

Name: `/par/out/elm/[oport]`
Access: `rw`
Type: `integer [degrees]`
Values: `[-90…90]`
Default: `5`

Measurements for satellites whose elevation angles are less than the specified mask won't be output to the `[oport]`.

Satellites Number Mask for Measurements Output

Name: `/par/out/minsvs/[oport]`
Access: `rw`
Type: `integer`
Values: `[0…255]`
Default: `0`

The receiver will not output measurements into the given `[oport]` as long as the number of satellites whose elevations exceed current elevation mask for measurements output is fewer than this parameter.

Output Epochs Counters

Name: `/par/out/epochs/[oport]`
Access: `r`
Type: `integer`
Default: `0`

This counter is incremented every time a new [~~](RT) message is output to the port `[oport]`. It is cleared every time the [~~](RT) message is being enabled to be output to the port, provided it is not already enabled.

Antenna Output Masks

Name: `/par/out/ant/[oport]`
Access: `rw`
Type: `array [1…4] of boolean`
Values: `{y|n,y|n,y|n,y|n}`
Default: `{y,n,n,n}`

Each element of the array enables output of observables taken from corresponding antenna to the port `[oport]`.

Note: This parameter is only available for multi-antenna receivers.

Antenna N Output Mask

Name: `/par/out/ant/[oport]/N (N=[1…4])`
Access: `rw`
Type: `boolean`
Values: `y,n`
Default: `y (N=0); n (N>0)`

This parameter enables output of observables taken from antenna `N` to the port `[oport]`.

Note: This parameter is only available for multi-antenna receivers.

Logging History

History logging provides statistical information on the raw data (receiver measurements) being logged to a selected stream. After *history logger* is associated with a specific output stream and data logging to this stream is enabled, *history logger* will start to collect and record corresponding information. It will record one bit of information per satellite every N seconds. This bit is set to unity if and only if at least some of the measurements have been stored in the last N seconds, and there have been no loss-of-lock events for the given satellite in the last N seconds. If either or both of these conditions are not met, a zero bit is recorded to the logging history.

The logging history has a limited capacity: a maximum of 32 satellites, 128 bits per satellite. Satellites for which all the bits are zero are not included in the logging history.

The information gathered by the logger could be obtained by means of the [LH] receiver message. For information about [LH] message, see “[LH] Logging History” on page 323.

Logging Period

Name: `/par/out/logh/period`
Access: `rw`
Type: `integer [seconds]`
Values: `[0…86400]`
Default: `30`

The history logging period. One bit per satellite is recorded every `period` seconds.

Output Stream to be Monitored

Name: `/par/out/logh/stream`
Access: `rw`
Type: `enumerated`
Values: `/[oport],/dev/null`
Default: `/dev/null`

The name of the output stream the history logger should gather information for. If the parameter is set to `/dev/null`, history logging will be disabled.

3.4.18 Parameters of Integrated GREIS Messages

These parameters allow you to tailor the integrated messages to your particular needs. For more information about the integrated messages, please see “Integrated Messages” on page 314.

Include CA/L1 Measurements into [rM] Message

Name: `/par/raw/rtm/meas/ca`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `on`

Include P1/L1 Measurements into [rM] Message

Name: `/par/raw/rtm/meas/p1`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `on`

Include P2/L2 Measurements into [rM] Message

Name: `/par/raw/rtm/meas/p2`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `on`

Version of Format of [rM] Message

Name: `/par/raw/rtm/ver`
Access: `rw`
Type: `integer`
Values: `0,1,2`
Default: `0`

`0` – the format is the default one.

`1` – the field `word2` (doppler) is excluded from the contents of the [rM] message.

`2` – the fields `flags`, `lock`, and `word2` are excluded from the contents of the [rM] message. In addition, `clock` field is reserved and contains zeroes.

Time Scale for Integrated Messages

Name: `/par/raw/rtm/tscale`
Access: `rw`
Type: `enumerated`
Values: `gps,glo,utc`
Default: `gps`

Type of Coordinates for [rV] Message

Name: `/par/raw/rtm/coord`
Access: `rw`
Type: `enumerated`
Values: `xyz,geo,grid`
Default: `xyz`

`xyz` – use Cartesian coordinates in [rV] message.

`geo` – use geodetic coordinates in [rV] message.

`grid` – use grid coordinates in [rV] message.

3.4.19 Parameters of NMEA messages

NMEA Standard Version

Name: `/par/nmea/ver`
Access: `rw`
Type: `enumerated`
Values: `v3.0,v2.3,v2.2`
Default: `v3.0`

This parameter instructs the receiver to generate NMEA messages according to the specified NMEA-0183 standard[12].

Offset for True Heading from HDT message

Name: `/par/nmea/head/offset`
Access: `rw`
Type: `float [degrees]`
Values: `(-360…360)`
Default: `0`

This parameter defines the direction with respect to which the True Heading will be calculated. The direction is defined as specified number of degrees clockwise from the True North.

Use "GP" as Talker ID in NMEA Messages

Name: `/par/nmea/gp`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `off`

12. NMEA-0183 Standard For Interfacing Marine Electronic Devices v.3.0. July 1, 2000.

`on` – always use “GP” as talker ID.

`off` – use “GP”, “GN”, or “GL” as talker ID according to the NMEA standard.

This parameter instructs the receiver to use “GP” as Talker ID in NMEA messages. This mode is implemented for compatibility with legacy equipment that may not be capable of recognizing “GN” or “GL” as Talker IDs.

Limit the Total Number of Satellites in GGA by 12

Name: `/par/nmea/ggalim`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `off`

`on` – no more than 12 satellites will be reported in GGA message.

`off` – actual number of satellites will be reported in GGA even when it exceeds 12.

In accordance with the NMEA-0183 standard, the total number of satellites in a GGA sentence is limited to 12. In practice, however, there may be more than 12 GPS satellites in sight. This parameter serves for compatibility with any software that strictly follows the NMEA-0183 standard.

Output Mode for HDT and ROT Messages

Name: `/par/nmea/head/fixed`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `off`

`on` – HDT and ROT messages will be output in RTK with fixed ambiguities position computation mode only.

`off` – HDT and ROT messages will be output with any position computation mode where corresponding values are calculated.

Enable GRS and GSA to Contain More Than 12 Satellites

Name: `/par/nmea/grsgsa`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `off`

`on` – GRS and GSA messages may contain more than 12 GNSS satellites.

`off` – GRS and GSA messages will never contain more than 12 satellites.

Mantissa Length for Fractional Seconds of UTC Time

Name: `/par/nmea/frac/sec`
Access: `rw`
Type: `integer`
Values: `[0…2]`
Default: `2`

This parameter specifies the number of digits in the fractional part of the seconds of UTC time.

Mantissa Length of Fractional Minutes for GGA Message

Name: `/par/nmea/frac/min/GGA`
Access: `rw`
Type: `integer`
Values: `[1…7]`
Default: `7`

This parameter specifies the length of mantissa for representation of factional minutes of latitude and longitude for GGA message.

Mantissa Length of Fractional Minutes for GLL Message

Name: `/par/nmea/frac/min/GLL`
Access: `rw`
Type: `integer`
Values: `[1…7]`
Default: `7`

This parameter specifies the length of mantissa for representation of factional minutes of latitude and longitude for GLL message.

Mantissa Length of Fractional Minutes for GNS Message

Name: `/par/nmea/frac/min/GNS`
Access: `rw`
Type: `integer`
Values: `[1…7]`
Default: `7`

This parameter specifies the length of mantissa for representation of factional minutes of latitude and longitude for GNS message.

Mantissa Length of Geoidal Separation and Orthometric Height

Name: `/par/nmea/frac/alt`
Access: `rw`
Type: `integer`
Values: `[1…4]`
Default: `4`

This parameter specifies the number of digits in the fractional meters for both geoidal separation and orthometric height (altitude above the geoid).

Mantissa Length of Fractional Degrees for VTG Message

Name: `/par/nmea/frac/deg/VTG`
Access: `rw`
Type: `integer`
Values: `[1…3]`
Default: `3`

Mantissa Length of Fractional Speed for VTG Message

Name: `/par/nmea/frac/speed/VTG`
Access: `rw`
Type: `integer`
Values: `[1…4]`
Default: `4`

Mantissa Length of Fractional Residuals for GRS Message

Name: `/par/nmea/frac/res/GRS`
Access: `rw`
Type: `integer`
Values: `[0…4]`
Default: `3`

Mantissa Length of Fractional Degrees for HDT and ROT Messages

Name: `/par/nmea/frac/deg/HDT`
Access: `rw`
Type: `integer`
Values: `[1…3]`
Default: `3`

3.4.20 Parameters of BINEX Messages

BINEX Site Name

Name: `/par/binex/site`
Access: `rw`
Type: `string[0…127]`
Values: `(arbitrary string)`
Default: `(empty string)`

The value of this parameter will be output into the field 0x04 of the BINEX record 0x00-00.

BINEX Data Identifier

Name: `/par/binex/data_id`
Access: `rw`
Type: `string[0…4]`
Values: `(arbitrary string)`
Default: `(empty string)`

The value of this parameter will be output into the field 0x0f of the BINEX record 0x00-00. If the length of the string is less than 4 characters, the value to be output to the field 0x0f will be padded on the right to 4 characters. The padding is performed using spaces.

Enable Fields of BINEX Record 0x00-00

Name: `/par/binex/00_00`
Type: `list {04,0f,17,19,1a,1b,1d,1f} of boolean`
Values: `{on|off,…,on|off}`
Default: `{on,…,on}`

Each element of this parameter enables output of corresponding field of BINEX record 0x00-00. When an element is `on`, the output of corresponding field is enabled, when an element is `off`, the output of corresponding field is disabled.

Note: To turn all the fields on or off, use `set,/par/binex/00_00/,on`, or `set,/par/binex/00_00/,off` command, respectively. Use separate field parameters described below to control separate fields.

Enable Field “F” of BINEX Record 0x00-00

Name: `/par/binex/00_00/F (F=[04,0f,17,19,1a,1b,1d,1f])`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `on`

3.4.21 File Management

Existing Files

List of Existing Files

Name: /log
Access: r
Type: list {…}
Default: {}

For each existing file, this list includes an entry called after the name of the file containing the file attributes. File attributes have the format {size,mtime} where:

size – the size of the file in bytes

mtime – the time of the last modification of the file in the format:

YYYYMMDDhhmmss

where YYYY is year, MM is month, DD is day, hh is hours, mm is minutes and ss is seconds.

Attributes of the File NAME

Name: /log/NAME
Access: r
Type: {size,mtime}

For an existing file NAME, this parameter contains its attributes (see description of the /log parameter). In addition to print and list, you can use remove commands with this parameter to remove the file NAME.

Size of the File NAME

Name: /log/NAME&size
Access: r
Type: integer [bytes]

Contents of the File NAME

Name: /log/NAME&content
Access: r

When you print this parameter, receiver will output raw contents of the file, i.e., not wrapped into [RE] messages. You can terminate the output any time by sending "#" character to the receiver. If you use command identifier, receiver will output [RE] message containing the identifier before the contents of the file.

Examples

Example: Print list of all the file attributes along with their names; remove one of the files; then print the list again:

```
⇒ print,/log:on
⇐ RE032/log={log1127b={      12910,2006Y11M27D13h21m25s},
  RE02D log1127a={      21465,2006Y11M27D13h21m12s}}
⇒ remove,/log/log1127b
⇒ print,/log:on
⇐ RE032/log={log1127a={      21465,2006Y11M27D13h21m12s}}
```

Example: List only the names of all existing files; remove all the files; list the names again:

```
⇒ list
⇐ RE013{log0113a,log1127a}
⇒ remove,/log/
⇒ list
⇐ RE002{}
```

Current Log-files

In this section, the notation `[a|b]` indicates the log-file `a` or log-file `b`, respectively; the user should specify either `a` or `b`.

File Name Prefix

Name: `/par/cmd/create/pre/[a|b]`
Access: `rw`
Type: `string[0…20]`
Default: `log`

This parameter determines the file name prefix used as part of automatically generated file names. File name is generated automatically when one creates a file using `create` command without arguments, or through Automatic File Rotation Mode (AFRM), or using the TriPad <FN> button to start data recording (see "*TriPad User's Manual*" at *http://www.javad.com*). You can set this parameter to a string comprising up to 20 characters valid for file names.

For valid characters, as well as for description of the algorithm used to automatically generate file names, please refer to the description of `create` command on page 41).

Note: `/par/cmd/create/pre/a` has a synonym, `/par/cmd/create/prefix`, that is provided for compatibility with older firmware versions that didn't have support for multiple log-files.

Current Log-file

Name: `/cur/file/[a|b]`
Access: `rw`
Type: `string`
Values: `(any existing file name)`
Default: `(empty string)`

These parameters contain the names of the current log-files, if such files exist, or empty strings, otherwise.

If there is no current log-file, then setting this parameter to an existing file's name will instruct the receiver to open this file for data appending thus making it the new "current" file.

If the current log-file exists, then changing this parameter will instruct the receiver to close the existing "current" file and then open a new "current" file.

If the command refers to a file that does not exist, the receiver will not create a new file and will not change the current log-file. Please refer to the description of `create` command on page 41 for the way to create new files and to make them the current log-files.

To stop data logging and close current log-file use the `dm` command described on page 37.

Note: `/cur/file/a` has a synonym, `/cur/log`, that is provided for compatibility with older firmware versions that didn't have support for multiple log-files.

Current Log-file Size

Name: `/cur/file/[a|b]&size`
Access: `r`
Type: `integer [bytes]`

The size of corresponding log-file, if any. An error message will be reported if there is no current log-file.

Note: `/cur/file/a&size` has a synonym, `/cur/log&size`, that is provided for compatibility with older firmware versions that didn't have support for multiple log-files.

Automatic File Rotation Mode (AFRM)

The JAVAD GNSS receiver has capability to rotate log-files automatically. The term *file rotation* means that the receiver closes the previous "current" file and opens a new one according to the user-defined schedule. The rotation schedule is specified by the two parameters called "file rotation period" and "file rotation phase."

File rotation is launched when the receiver time modulo “`period`” is equal to “`phase`”. More precisely, a new log-file is opened immediately before the scheduled epoch so that all of the data tagged with this epoch will be recorded into the new log-file.

In addition, AFRM uses a counter that is decremented on every file rotation event until its value becomes zero. Once the counter is zero, file rotation automatically stops. This feature allows to create as many log-files as necessary and then stop data logging. The counter is initialized simultaneously with AFRM, and its initial value is set through the parameter `/par/log/rot/sc/count` (see below).

Note that a log-file is opened right after turning AFRM on. The opening of such a “start-up” file, however, is not considered a file rotation event and, therefore, the AFRM counter will not be decremented.

When opening a new log-file, the receiver enables the default set of messages outputted with the default output period. Both the default set of messages and the default output period are programmable.

The JAVAD GNSS receiver is capable of removing the “oldest” log-files, if there is no free memory left to continue data logging. This feature is off by default. Even if you turn it on, your receiver will not delete the files with the earliest file creation times unless AFRM is also on. The latter condition is essential since it allows you to minimize the risk of inadvertent file deletion.

Log-files Management Parameters

Name: `/par/log`
Access: `r`
Type: `list {rot,imp,a,b}`

This set of parameters defines the rules for automatic and implicit management of the receiver log-files. The automatic and implicit management is performed by automatic file rotation mode (AFRM), and by TriPad interface. *Implicit* in this context means that file management in these cases is performed not through regular GREIS commands, but by internal receiver algorithms.

`rot` – automatic log-files rotation mode (AFRM) parameters.

`imp` – enable implicit management of current log-files

`a` – implicit output parameters for `/cur/file/a`

`b` – implicit output parameters for `/cur/file/b`

File Rotation Parameters

Name: `/par/log/rot`
Access: `r`
Type: `list {rmold,mode,force,count,sc}`

Enable Oldest Log-file Removal

Name: `/par/log/rot/rmold`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `off`

`on` – enable automatic removal of the oldest log-file while AFRM is on and there is not enough free space for files to continue data logging. It doesn't enable automatic removal when AFRM is not active.

`off` – disable automatic removal of files.

File Rotation Mode

Name: `/par/log/rot/mode`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `off`

`on` – enable AFRM.

`off` – disable AFRM.

Turning this parameter from `off` to `on` starts data logging to the log-files enabled by the `/par/log/imp` parameter and copies the value of the `/par/log/rot/sc/count` parameter to `/par/log/rot/count`.

The names of the created files are generated automatically as if the `create` command without arguments has been issued. The default set of messages, `/msg/def`, is enabled to be output to the file(s) with the period specified by corresponding `/par/log/[a|b]/sc/period` parameter.

When `on`, every time the receiver time matches the equation T_r(mod `period`)=`phase`, where `period` and `phase` are the values of `/par/log/rot/sc/period` and `/par/log/rot/sc/phase`, respectively, the *rotation event* is generated. The rotation event causes current log-file(s) to be closed, and new log-file(s) to be created. More precisely, new log-file(s) is opened immediately before the scheduled epoch so that all of the data tagged with this epoch will be recorded into the new log-file(s).

In addition, every rotation event will decrement the value of the /par/log/rot/count parameter provided its value is greater than zero. Should the count drop to zero, the AFRM is automatically turned off. The initial creation of file(s) when AFRM is being turned on is *not* considered to be a rotation event.

Turning this parameter from on to off closes the log-files enabled by the /par/log/imp parameter.

Force File Rotation

Name: /par/log/rot/force
Access: rw
Type: boolean
Values: on, off
Default: off

on – provided AFRM is turned on, will force the *rotation event* to be generated at the closest internal receiver epoch.

off – has no effect.

File Rotation Running Counter

Name: /par/log/rot/count
Access: r
Type: integer
Values: [0…2^{31}-1]

This parameter indicates how many *rotation events* remains to happen before the file rotation stops. Zero means an unlimited number of files.

Refer to /par/log/rot/sc/count below for the method of changing this parameter.

File Rotation Scheduling Parameters

Name: /par/log/rot/sc
Access: r
Type: list {period,phase,count}

When AFRM is on, every time the receiver time matches the equation T_r (mod period) = phase, the *rotation event* is generated.

Every rotation event will decrement the value of the count field provided its value is greater than zero. Should the count drop to zero, the AFRM is automatically turned off.

File Rotation Period

Name: /par/log/rot/sc/period
Access: rw
Type: float [seconds]
Values: [60…86400]
Default: 3600

File Rotation Phase

Name: /par/log/rot/sc/phase
Access: rw
Type: float [seconds]
Values: [0…86400]
Default: 0

File rotation counter

Name: /par/log/rot/sc/count
Access: rw
Type: integer
Values: [0…231-1]
Default: 0

This parameter specifies the total number of files that will be created before file rotation is turned off, 0 meaning unlimited number of files. The value of this parameter is copied to `/par/log/rot/count` whenever `/par/log/rot/mode` is turned from `off` to `on`. In addition, when `/par/log/rot/mode` is `on`, setting this parameter will update the value of `/par/log/rot/count`.

Enable Implicit Management of Current Log-files

Name: /par/log/imp
Access: rw
Type: array [0…1] of boolean
Values: {y|n,y|n}
Default: {y,n}

The first element of the array corresponds to the `/cur/file/a`, and the second element – to the `/cur/file/b`.

Enable Implicit Management of Specific Current Log-file

Name: /par/log/imp/N (N=0,1)
Access: rw
Type: boolean
Values: y,n
Default: y for N=0, n for N=1

N=0 corresponds to the /cur/file/a, and N=1 – to the /cur/file/b.

y – corresponding current log-file will be controllable via AFRM and TriPad.

n – corresponding current log-file will be ignored by AFRM and TriPad.

Implicit Message Output Period

Name: /par/log/[a|b]/sc/period
Access: rw
Type: float [seconds]
Values: [0…86400]
Default: 1

This parameter specifies the interval of outputting messages into the log-file when data logging is activated with the TriPad or through the AFRM.

Internal Disk Parameters

Blocks Count

Name: /dev/blk/a&blocks
Access: r
Type: integer

Number of blocks on the internal block device. The internal block device is used by the receiver file-system to store receiver files.

Block Size

Name: /dev/blk/a&block_size
Access: r
Type: integer [bytes]

The size of a single block on the internal block device.

File-system Parameters

Available Memory

Name: /par/dev/blk/a/size
Access: r
Type: integer [bytes]

Free Memory

Name: `/par/dev/blk/a/free`
Access: `r`
Type: `integer [bytes]`

Block Size

Name: `/par/dev/blk/a/block_size`
Access: `r`
Type: `integer [bytes]`

Maximum Number of Files

Name: `/par/dev/blk/a/max_files`
Access: `r`
Type: `integer`

The file-system will refuse to create a new file if the current number of files on the file-system is greater or equal to the value of this parameter.

Number of Files

Name: `/par/dev/blk/a/files`
Access: `r`
Type: `integer`

Current number of files on the file-system.

Number of Bad Blocks

Name: `/par/dev/blk/a/bad_blocks`
Access: `r`
Type: `integer [bytes]`

Verification of Writing

Name: `/par/dev/blk/a/verify`
Access: `rw`
Type: `enumerated`
Values: `off,fast,slow`
Default: `fast`

File-system Version

Name: `/par/dev/blk/a/ver`
Access: `r`
Type: `integer`

The version of file-system the block device was formatted by.

Format Time

Name: `/par/dev/blk/a/tfmt`
Access: `r`
Type: `integer [seconds]`

The time-tag of the last initialization (format) of the file system. The time is measured in seconds since 00:00:00 Jan 1, 1986.

Amount of Data

Name: `/par/dev/blk/a/data`
Access: `r`
Type: `integer`

Number of bytes of data stored on the file-system. This number is usually less than number of used blocks multiplied by block size as every file may waste up to one block of data due to the file-system requirement that every file occupies integer number of blocks.

The value of this parameter is what the _MEM option is limiting since firmware version 2.5p2, instead of the old behavior, when _MEM limited the number of used blocks multiplied by block size.

Show Removed Files

Name: `/par/dev/blk/a/removed`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `off`

`off` – receiver file system behaves as usual

`on` – receiver file system allows to view only removed files, and is in read-only mode.

This parameter allows the user to recover inadvertently deleted files.

The receiver's file system will be remounted every time this parameter is set to either `on` or `off`. The alternative way to force the file system to remount is clearing the NVRAM, but this may be unacceptable in many cases.

Note: The command `init,/par/` won't set this parameter to the default `off` value.

Note: This parameter is not stored in the receiver's NVRAM, so it will always be set to `off` after the receiver is powered on.

File-system Initialization Progress

Name: `/par/stat/fsinit`
Access: `r`
Type: `list {total,processed}`

These two fields allow monitoring of the file system initialization or remount progress. When initialization or remount is not in progress, these two values are the same.

`total` – the total number of blocks used for file storage.

`processed` – the number of blocks that are already processed.

File System Buffer Allocation

Name: `/par/stat/fsb`
Access: `r`
Type: `list {sz,max,cnt}`

This parameter is intended for the JAVAD GNSS's firmware developers and is subject to change at any time.

`sz` – the size of the file system buffer.

`max` – maximum number of bytes in the file system buffer since the last receiver start-up.

`cnt` – current number of bytes in the file system buffer.

3.4.22 Session programming

Overview

Session programming means specifying one or more *jobs* for *session scheduler*. Basically, a job is a set of receiver commands executed at the specified time(s). The session scheduler can handle a fixed number of jobs. Each job has a unique integer identifier, counting from zero. Each job specification comprises:

`spec` – time specification

`cmds` – index of the command set to be executed

`count` – counter

`port` – output port

Besides the above mentioned four fields, every job has an *activity flag*. Jobs whose activity flags are not set, will be ignored by the session scheduler. A job with the activity flag set to unity is called *active job*. An active job will be executed by the session scheduler as soon as the current time matches the value of the `spec` field. If two or more jobs

are programmed to be executed at the same time, the jobs will be executed in the order specified in the scheduler, e.g., if the jobs with identifiers `0` and `1` are to be started at the same time, job `0` will be executed before job `1`.

When it's time to execute a job, the scheduler takes index from the `cmds` field of the job specification and executes command(s) found at this index in the array of command strings. Using index to identify command string allows multiple jobs to share the same command string. The string found is executed the same way if it were received through input port specified in the `port` field of the job. As a consequence of this rule, should command generate some output, the output is sent to the `port`. Current input mode of the `port` doesn't matter though, – the command is executed as if the port is in the command mode anyway.

Every active job also serves as a wake-up point for the receiver. Therefore if session scheduler is active and receiver is in sleep mode, the next job to be executed will first wake-up receiver and only then corresponding commands will be executed. This feature also makes it useful to have empty command string as a job specification. Such specification will just wake-up receiver without execution of any commands.

When session scheduler is being turned on or is being restarted, the scheduler makes a copy of *activity flag* and `count` field of every job. Let's call these copies `stat/active` and `stat/count`, respectively. Every time the job is executed and `stat/count` is non-zero, the `stat/count` is decremented. Should `stat/count` become zero as a result of decrement, the `stat/active` flag is turned off. The job is thus deactivated and stays inactive until scheduler mode is changed or scheduler is restarted. This allows to limit number of executions of a job by setting its `count` field to a non-zero value. The `stat/mode` and `stat/count` fields of every job could be read from the receiver for reference but couldn't be directly changed by user.

Time specification `spec` is a template containing four fields, namely day of week `day`, hour of day `hour`, minute of hour `minute`, and second of minute `second`. Each field can either be set to an integer value in corresponding range, or to a special value that serves as a wild-card that matches any value. In order for a job to be executed at given time `T`, job's `spec` should match given time `T`. Scheduler compares current time against `spec` by first decomposing current time into the set of corresponding fields, then comparing every field of decomposed time against corresponding field of `spec`. Time matches `spec` if and only if every field of time matches corresponding field of `spec`. Fields of `spec` having special value match any value of corresponding field of time. For example, specification `2d17h__m30s`, where "__" (double underscore) is special value, matches `2:17:00:30`, `2:17:01:30`,…, `2:17:59:30`. It means that job having such `spec` will be executed at the middle of every minute during one hour starting on Monday, 17:00:00. Due to the fact that week number couldn't be specified, the job will in fact be executed every Monday at 17:00:30, 17:01:30,…, and 17:59:30 (60 times). To limit executions of

such job to single (next) Monday, job counter should be set to 60 before activation of the session.

In fact session scheduler doesn't check `spec` of every active job against current time every second. Instead it determines job that should be executed next (along with its next execution time) whenever scheduler mode is changed, scheduler is restarted, or job is executed. The identifier of the next job to be run and corresponding execution time along with current time could be read from the receiver.

Parameters

Session Mode

Name: `/par/sess/mode`
Access: `rw`
Type: `enumerated`
Values: `on,off,susp`
Default: `off`

This parameter specifies the current session scheduler mode.

`off` – session scheduler is disabled

`on` – session scheduler is active

`susp` – session scheduler is active and works almost as usual, but does not actually execute commands.

Restart Session Scheduler

Name: `/par/sess/restart`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `off`

`on` – restarts session scheduler. This is similar to setting `/par/sess/mode` to `off` then back to `on`. The value of the parameter is returned back to `off` after scheduler restart.

`off` – ignored.

Job Activity Flags

Name: `/par/sess/active`
Access: `rw`
Type: `array [0…15] of boolean`
Values: `{y|n,…,y|n}`
Default: `{n,…,n}`

The value of this parameter is copied to the `/par/sess/stat/active` one whenever session scheduler mode is changed or scheduler is restarted.

Job #N Activity Flag

Name: `/par/sess/active/N (N=[0…15])`
Access: `rw`
Type: `boolean`
Values: `y,n`
Default: `n`

`y` – job is enabled.

`n` – job is disabled.

Job Specifications

Name: `/par/sess/job`
Access: `rw`
Type: `array [0…15] of job specification`

Job #N Specification

Name: `/par/sess/job/N (N=[0…15])`
Access: `rw`
Type: `list {spec,cmds,count,port}`
Default: `{__d__h__m__s,0,0,""}`

Job #N Time Specification

Name: `/par/sess/job/N/spec (N=[0…15])`
Access: `rw`
Type: `timespec`
Default: `__d__h__m__s`

Job is executed whenever current time matches its time specification. "__" in time specification matches any value.

Job #N Index of Command String

Name: `/par/sess/job/N/cmds (N=[0…15])`
Access: `rw`
Type: `integer`
Values: `[0…7]`
Default: `0`

This parameter designates index into the array `/par/sess/cmds`, which corresponding element in turn contains the command(s) to be executed.

Job #N Execution Counter

Name: `/par/sess/job/N/count (N=[0…15])`
Access: `rw`
Type: `integer`
Values: `[0…2147483647]`
Default: `0`

If non-zero, designates number of times the job should be executed by the scheduler before deactivation. If zero, execution is not limited.

The value is copied to the corresponding `/par/sess/stat/count/N` parameter every time session scheduler mode is changed or scheduler is restarted.

Job #N Current Terminal

Name: `/par/sess/job/N/port (N=[0…15])`
Access: `rw`
Type: `enumerated`
Values: `/[oport],(empty string)`
Default: `(empty string)`

The commands assigned to the job are executed as if they were received from the port specified by this parameter. Should the commands generate receiver replies, they will be sent to corresponding output port.

Command Strings

Name: `/par/sess/cmds`
Access: `rw`
Type: `array [0…15] of strings`
Default: `{"",…,""}`

Command String #N

Name: `/par/sess/cmds/N, where N= 0…15`
Access: `rw`
Type: `string [0…64]`
Default: `"" (empty string)`

Session Scheduler Status

Name: `/par/sess/stat`
Access: `r`
Type: `list {active,count,job,time}`

Running Activity Flags

Name: `/par/sess/stat/active`
Access: `r`
Type: `array [0…15] of boolean`
Values: `{y|n,…,y|n}`

This parameter is set to the value of `/par/sess/active` parameter whenever session scheduler mode is changed or session scheduler is restarted.

Running Activity Flag for Job #N

Name: `/par/sess/stat/active/N (N=[0…15])`
Access: `r`
Type: `boolean`
Values: `y,n`

Job Execution Down-counters

Name: `/par/sess/stat/count`
Access: `r`
Type: `array [0…15] of integer`

Job #N Execution Down-counter

Name: `/par/sess/stat/count/N (N=[0…15])`
Access: `r`
Type: `integer`
Values: `[0…2147483647]`

Whenever session scheduler mode is changed or scheduler is restarted, the value of corresponding `/par/sess/job/N/count` parameter is copied to this variable. If the variable is greater than zero, it is decremented every time corresponding job is executed. Should the variable became zero as a result of decrement, corresponding current activity flag `/par/sess/stat/active/N` is set to “`n`”, thus the job is deactivated.

Next Job to be Executed

Name: `/par/sess/stat/job`
Access: `r`
Type: `integer`
Values: `[-1…15]`
Default: `-1`

`-1` – there are no jobs to be executed

`[0…15]` – job index to be executed next

Timing Information

Name: /par/sess/stat/time
Access: r
Type: list {next,curr,delta}

Time of Execution of Next Job

Name: /par/sess/stat/time/next
Access: r
Type: timespec

If there is no next job, the value will be `__d__h__m__s`.

Current Time

Name: /par/sess/stat/time/curr
Access: r
Type: timespec

Current time in timespec format that is running no matter what scheduler mode is.

Time Left to Next Job Execution

Name: /par/sess/stat/time/delta
Access: r
Type: timespec

If there is no next job, the value will be `__d__h__m__s`.

Examples

Example: Suppose we need to program receiver for a single session that will begin next Wednesday, 12:30:00 and end on Thursday, 10:00:00. During this time receiver should write file "ses.log" containing the default set of messages into its internal memory at 1Hz and simultaneously output NMEA GGA message to its serial port B at 10Hz. Except this time period, receiver should be turned off. In fact there are multiple ways to achieve this, here is one of them. Let's define two jobs for the session:

```
job 0:
  spec  = 4d12h30m00s
  cmd   = "create,ses.log;em,/cur/log,def:1;em,/dev/ser/b,nmea/GGA:0.1"
  count = 1

job 1:
  spec  = 5d10h00m00s
  cmd   = "dm,/dev/ser/b;dm,/cur/log;set,power,off"
  count = 1
```

After programming these two jobs and turning session scheduler on, we can put receiver into sleep mode (using MINTER or `set,sleep,on` command). Receiver will wake-up at the time specified by the job `a` and execute corresponding commands. When the time for the job `b` comes, receiver will execute corresponding commands turning receiver off as a result. Note that we turn power off in the job `b` as opposed to putting receiver into sleep mode. This way we don't need to set counters for jobs because once receiver is turned off, it can't wake-up anymore. This however, has a side effect that both jobs will remain active and thus may trigger corresponding actions next time we turn receiver power on. If we don't want that, we will need to set `count` field for both jobs to 1 while programming.

Note also that if we activate such session after Wednesday, 12:30:00 but before Thursday, 10:00:00, the receiver will not do what we meant. The first job that will be run in this case is job `b`, not job `a`, so job `a` won't be executed at all.

Here are actual commands to program the above jobs (recall that '#' is comment character):

```
⇒ # Turn off scheduler and deactivate all jobs
  set,/par/sess/mode,off
  set,/par/sess/active,n
  # Define 'spec', 'cmds', 'count', and 'port' fields for jobs 0 and 1
  set,/par/sess/job/0,{4d12h30m0s,0,1,/dev/null}
  set,/par/sess/job/1,{5d10h0m0s,1,1,/dev/null}
  # Define commands 0 and 1
  set,/par/sess/cmds/0,"create,ses.log;em,/cur/log,def:1;em,/dev/ser/b,
    nmea/GA:0.1"
  set,/par/sess/cmds/1,"dm,/dev/ser/b;dm,/cur/log;set,power,off"
  # Activate jobs 0 and 1
  set,/par/sess/active,{y,y}
  # Turn scheduler on
  set,/par/sess/mode,on
```

●

Example: Suppose we need to setup a permanent station that will work from 8:00:00 to 20:00:00 every day from Monday to Friday and will be turned off during weekend (Saturday and Sunday). During work-time the station should transmit differential correction through one of its serial ports and write measurement files to its internal memory. The files should be automatically rotated every hour and oldest file should be deleted when there is not enough memory to write latest data.

To achieve this we first program receiver to be base station to send corrections and configure AFRM mode to handle logging stuff (not shown in this example).

Then program the following 4 jobs:

```
job 0: spec = _d08h00m00s cmd = ""
job 1: spec = _d20h00m00s cmd = "set,sleep,on"
job 2: spec = 7d08h00m00s cmd = "set,sleep,on"
job 3: spec = 0d08h00m00s cmd = "set,sleep,on"
```

Jobs `a` and `b` will turn receiver on and off at specified times every day. But that is not exactly what we want as receiver then will work on Saturday and Sunday as well. To prevent this, we configure jobs `c` and `d` that will put receiver into sleep mode immediately after wake-up on Saturday and Sunday, respectively. There will be short periods of time when receiver is turned on Saturday and Sunday, but who cares? Note that we don't need jobs similar to `c` and `d` to be run at 20:00:00 on Saturday and Sunday as job `b` will put receiver into sleep mode anyway.

Here are the commands (base station configuration and AFRM programming not shown):

```
⇒ # Turn off scheduler and deactivate all jobs
  set,/par/sess/mode,off
  set,/par/sess/active,n
  # Define 'spec', 'cmds', 'count', and 'port' fields for jobs 0…3
  set,/par/sess/job/0,{  8h0m0s,0,0,/dev/null}
  set,/par/sess/job/1,{ 20h0m0s,1,0,/dev/null}
  set,/par/sess/job/2,{7d8h0m0s,1,0,/dev/null}
  set,/par/sess/job/3,{0d8h0m0s,1,0,/dev/null}
  # Define commands 0 and 1
  set,/par/sess/cmds/0,""
  set,/par/sess/cmds/1,"set,sleep,on"
  # Activate jobs
  set,/par/sess/active,{y,y,y,y}
  # Turn scheduler on
  set,/par/sess/mode,on
```

●

Example: Suppose we need to output the size of current log-file to receiver port A every second. There is no predefined message that contains this information, but there is corresponding parameter that we can print, so we program single job to achieve the required result:

```
job a:
  spec = _d_h_m_s
  cmd  = "%job_a : size=%print,/cur/file/a&size"
  port = /dev/ser/a
```

When this job is active, receiver will output RE message to the port A every second in the form:

```
⇐ REXXX%job_a : size=% SIZE
```

where `SIZE` is current file size. We've put "`%job_a : size=%`" into the command to be able to identify the reply got from the job.

Here are actual commands:

```
⇒ # Turn off scheduler and deactivate all jobs
  set,/par/sess/mode,off
  set,/par/sess/active,n
  # Define 'spec', 'cmds', 'count', and 'port' fields for job 0
  set,/par/sess/job/0,{"",0,0,/dev/ser/a}
  # Define command 0
  set,/par/sess/cmds/0,"%job_a : size=%print,/cur/file/a&size"
  # Activate job 0
  set,/par/sess/active/0,y
  # Turn scheduler on
  set,/par/sess/mode,on
```

Note: This example demonstrates one use of the `port` field of job specification. Another one would be monitoring of execution of session commands. By setting `port` to one of receiver ports it is possible, e.g., to see if any errors occur as a result of job execution.

3.4.23 Notebook

Notebook allows the user to store arbitrary information into the receiver and retrieve it later if necessary. In addition, as this information is output in the [PM] message along with other parameters, it could be used to pass arbitrary text data from controller application to a post-processing one[13].

Notes

Name: `/par/note`
Access: `r`
Type: `list {str,nv}`

13. See also "event" on page 46 for another way to communicate to post-processing applications.

Note Strings

Name: /par/note/str
Access: r
Type: array[0…7] of strings

Note String #N

Name: /par/note/str/N (N=[0…7])
Access: rw
Type: string [0…64]

When setting a string for index N, the corresponding flag /par/note/nv/N is cleared (set to n) to indicate that this string is set by the user, not read from the NVRAM.

Change Indicators

Name: /par/note/nv
Access: r
Type: array [0…7] of boolean

When receiver starts up, it reads all the note strings from its NVRAM and sets all the values in this array to y. When user sets some of strings, corresponding NVRAM flag is cleared (set to n).

3.4.24 Generic Communication Parameters

The parameters described in this section are common for all the supported ports.

Current Terminal

Current Terminal

Name: /cur/term
Access: r
Type: string
Values: /[port]

The name of the current terminal, i.e., the name of the input stream the command requesting the value was received through.

Basic Operation Modes

Input Mode

Name: `/par/[port]/imode`
Access: `rw`
Values: `cmd,echo,jps,rtcm,rtcm3,cmr,omni,none,dtp,term`
Default: `cmd`

With this parameter the user specifies what type of incoming data to accept on the selected receiver port.

`cmd` – command mode. Being in this mode, the receiver's port recognizes GREIS commands sent by the user.

`echo` – echo mode. This mode is the same as `cmd` mode unless `/par/[port]/echo` is set to a value different from `/dev/null`, see `/par/[port]/echo` parameter description below for details.

`jps` – GREIS input mode. In this mode receiver is capable to recognize both standard and non-standard GREIS messages.

`rtcm` – RTCM 2.x input mode.

`rtcm3` – RTCM 3.x input mode. In this mode the receiver recognizes and decodes the RTCM 3.x messages received through the corresponding port.

`cmr` – CMR/CMR+ input mode. For more information on CMR format, please refer to ftp://ftp.trimble.com/pub/survey/cmr.

`omni` – unsupported.

`none` – means that the port will ignore any incoming data.

`dtp` – the port is currently attached to the Data Transfer Protocol (DTP), so all the input goes there. This mode could be set only by the `get` GREIS command. The mode will return to `cmd` as soon as DTP terminates.

`term` – the PPP data link is currently established over this port, so all the input goes there. This mode could be set only implicitly by the PPP stack. When parameter is implicitly set to this mode, attempts to change the mode will fail.

Output Mode

Name: `/par/[port]/omode`
Access: `r`
Values: `std,dtp,term`
Default: `std`

`std` – standard output mode.

`dtp` – GREIS Data Transfer Protocol (DTP) is active on the port. This mode could be set only by the `get` GREIS command. The mode will return to `std` as soon as DTP terminates. In this mode all the usual messages output to the given port is temporarily suppressed.

`term` – the PPP data link is currently established over this port. The mode will return to `std` as soon as PPP terminates. In this mode all the usual messages output to the given port is temporarily suppressed.

Output Duplication

Name:	`/par/[port]/dup`
Access:	`rw`
Type:	`enumerated`
Values:	`/[oport],/dev/null`
Default:	`/dev/null`

This parameter specifies an output port to which all the data being output to the `[port]` should be sent (duplicated).

`/[oport]` – the outgoing data will be sent (duplicated) to specified output port. The outgoing data that will be duplicated include regular message output, the data echoed (see "Echo Parameters" below) from another port, and the data duplicated from another port; if any.

`/dev/null` – no data duplication will be performed.

Besides troubleshooting, this feature is useful if you need to request the output of exactly the same messages into multiple ports. Instead of programming each port output individually, the messages could be enabled to one of the ports, and then duplicated to another one, and then duplicated from those one to yet another one, etc. Not only it's simpler to program, but it will also reduce processor load as CPU won't spend time to re-generate the same messages multiple times.

Echo Parameters

Echo Port

Name:	`/par/[port]/echo`
Access:	`rw`
Type:	`enumerated`
Values:	`/[oport],/dev/null`
Default:	`/dev/null`

This parameter specifies an output port to which all the incoming data being received from the `[port]` should be sent.

/[oport] - the incoming data will be sent (echoed) to specified output port. Echoing the data doesn't prevent interpretation of them according to currently selected input mode, unless the input mode is set to echo, in which case the data will be otherwise ignored.

/dev/null - the incoming data won't be sent to any of output ports.

Given the above description, to achieve just echoing of the data from an input port to an output one (*pure echo*), one needs to set the input mode to echo, and the echo port to the name of the required output port. If the input port that is to be turned into the pure echo mode is the current port, the sequence of commands shown in the examples below is recommended.

Note: To program a feature sometimes referred to as "daisy-chain", i.e., bidirectional virtual channel between two ports, it's required to set pure echo mode on both ports participating in the daisy chain.

Echo-off Sequence

Name:	/par/[port]/eoff
Access:	rw
Type:	string [4…32]
Values:	(arbitrary string)
Default:	#OFF#

The sequence of characters that will reset /par/[port]/echo to its default /dev/null value once it is received through the [port].

Note that receiving of the echo-off sequence doesn't change the input mode of the current port. Instead, to conveniently support turning from pure echo to command mode, the echo input mode behaves exactly like cmd input mode when /par/[port]/echo parameter is set to /dev/null.

Note: The default value for this parameter is intentionally chosen so that it will be considered to be just a comment by the GREIS language parser. It makes it safe to send the default echo-off sequence even when corresponding port is currently in command mode.

Warning: *The echo-off sequence will be echoed to the current echo port before the current echo port is turned to /dev/null. If you setup a daisy-chain between your controller and some device connected to receiver port, make sure to first program echo-off sequence to a value that will do no harm to the device. On the other hand, don't change the echo-off sequence from its default value without necessity, as other applications trying to establish communication with the receive will likely to fail provided your leave the port in the pure echo mode.*

Enable Wrapped Echo

Name: /par/[port]/ewrap
Access: rw
Type: boolean
Values: on,off
Default: off

on – when this parameter is on and /par/[port]/echo is other than /dev/null, data echoing is carried out in the “wrapped” mode, i.e. all of the characters received from [port] are combined into a corresponding [>>] message (see “[>>] Wrapper” on page 322) before being output to the output port.

off – no wrapping of the data will occur.

For the purposes of wrapping, the data are stored in the internal receiver buffer until either timeout in the receiving of data occurs, or the amount of data in the buffer exceeds the wrapping threshold. The timeout is currently not customizable and set to 100 milliseconds, and the threshold could be changed by the /par/[port]/wsize parameter.

Wrapping Threshold

Name: /par/[port]/wsize
Access: rw
Type: integer
Values: [1…128], bytes
Default: 128

This parameter specifies the threshold value for wrapping input data when in the wrapped mode.

Examples

Example: Setup pure echo from the current port to the serial port B. The commands below will do the job even if current port is already in the echo mode to the same or some other port, provided echo off sequence was not changed from its default value. The commands also make sure none of them are echoed to the destination port:

⇒ #OFF#

⇒ set,/cur/term/imode,cmd

⇒ set,/cur/term/echo,/dev/null

⇒ set,/cur/term/imode,echo

⇒ set,/cur/term/echo,/dev/ser/b

Example: Setup *daisy chain* between current port and serial port B. The commands below will do the job even if current port is already in the echo mode to the same or some other port, provided echo off sequence was not changed from its default value:

```
⇒ #OFF#
⇒ set,/cur/term/imode,cmd
⇒ set,/cur/term/echo,/dev/null
⇒ set,/par/dev/ser/b/imode,echo
⇒ set,/par/dev/ser/b/echo,/cur/term
⇒ set,/cur/term/echo,/dev/ser/b
⇒ set,/cur/term/imode,echo
```

●

Advanced Input Mode

Overview

JAVAD GNSS receivers support advanced input mode. This mode allows to use single input port (being set to `jps` input mode) to feed receiver with data in multiple formats, as well as dispatch different data to different decoders. In this mode the receiver will decode the [>>] messages and non-standard messages and will pass the data decoded from these messages to a specified decoder (e.g., commands interpreter or RTCM decoder), or will send the decoded data to a specified output port. To provide backward compatibility with the earlier firmware versions, advanced input mode is turned off by default. In this case the [>>] messages and non-standard messages are ignored when the input mode is set to `jps`.

Note: Remember that the primary purpose of the `jps` input mode is to receive and decode those messages in GREIS format that carry information suitable for phase-differential mode of position computation.

There is a set of parameters through which advanced input mode can be controlled. This set is represented by an array of three[14] elements. Each element of the array is a *specification* that consists of the following fields: `mode`, `id`, `skip`, and `port`. When either [>>] or non-standard message is received, its contents is checked against every specification in turn according to the rules described below. The first specification that matches will govern the execution of the message contents. If no matching specification is found, the contents of the message is ignored. In the *matching stage*, the receiver uses the `mode` and `id` fields. In the *execution stage*, the receiver uses `mode`, `skip`, and `port` fields.

14. The number of elements is somewhat arbitrary and can be expanded in the future (if required).

Matching stage

On this stage, the receiver compares the `mode` and `id` fields with the contents of the received message.

Depending on the value of the `mode` field, the specification is allowed to match a message as follows:

`none` – the specification never matches any message.

`cmd, echo, jps, rtcm, rtcm3, cmr` – the specification could match the [>>] message, but never matches a non-standard message.

`nscmd, nsecho` – the specification could match a non-standard message but never matches the [>>] message.

Once the value in the `mode` field allows to match given message, the receiver will compare the value in the `id` field of the specification with the *identifier* of the message. For [>>] message, its identifier is the value of its `id` field. For the non-standard messages, the first byte of the message is taken as its identifier. The message matches the specification in two cases:

1. The value of the message identifier exactly matches the value of the `id` field of specification.
2. The value of the message identifier doesn't matter if the value `id` field of the specification has a special value `-1`.

Execution stage

Once the message passes the matching stage, i.e., there is a specification that matches the message, the contents of the message is executed according to the first matching specification as follows:

If the `skip` field is set to `y`, the first byte of the message is skipped before executing the message contents. Otherwise the first byte is considered as the part of the message contents. Note that for [>>] message the contents never includes its generic header, i.e., the contents is its body.

After optional skipping of the first byte, the message contents is executed according to the value of the `mode` field in the matched specification as follows:

`jps, rtcm, rtcm3, cmr` – the receiver will pass the contents to the corresponding type of decoder as if the message has been received through the port specified in the `port` field of the specification. The input mode of the corresponding port (`/par/[port]/imode`) should be set accordingly for execution to actually take place. The empty value of the `port` field denotes the current port (i.e., the port the initial message is received through). Bear in mind that the empty value of the `port` field

for `jps` mode is meaningless (as well as explicit setting the port parameter to the port matching those the parameter is being set for) as the contents would pass to the same decoder the initial message came from. This, in turn, would break the decoder logic, so the receiver will protect itself by just throwing away the contents of the message.

`cmd`, `nscmd` – the receiver will pass the contents to the command interpreter as if the command has been received through the port `port`. The empty value of `port` denotes the current port.

`echo`, `nsecho` – the receiver will either send the contents to the output port specified in the `port` field of the matched specification.

Parameters Description

Input Specifications

Name: `/par/[port]/jps`
Access: `rw`
Type: `array [0…2] of input specification`

Input Specification

Name: `/par/[port]/jps/N (N=[0…2])`
Access: `rw`
Type: `list {mode, id, skip, port}`

Matching/Execution Mode

Name: `/par/[port]/jps/N/mode (N=[0…2])`
Access: `rw`
Type: `enumerated`
Values: `none,cmd,echo,jps,rtcm,rtcm3,cmr,nscmd,nsecho`
Default: `none`

In the *matching stage*, the values have the following meaning:

`none` – the specification will never match any message.

`cmd`, `echo`, `jps`, `rtcm`, `rtcm3`, `cmr` – the specification may possibly match the [>>] message. The matching criterion is the product of comparison between the `id` field of the [>>] message and the `id` field of the specification.

`nscmd`, `nsecho` – the specification may possibly match a non-standard message. The matching criterion is the product of comparison between the first byte of the non-standard message and the `id` field of the specification.

In the *execution stage*, the values have the following meaning:

`none` – never appears in the execution stage as no message could match this mode.

`cmd` – execute the message contents as command(s) received from the port specified in the `port` field of the input specification.

`echo` – send the message contents to the port specified in the `port` field of the input specification.

`jps` – process the message contents as data in GREIS format received from the port specified in the `port` field of the input specification.

`rtcm` – process the message contents as data in RTCM 2.x format received from the port specified in the `port` field of the input specification.

`rtcm3` – process the message contents as data in RTCM 3.0 format received from the port specified in the `port` field of the input specification.

`cmr` – process the message contents as data in CMR format received from the port specified in the `port` field of the input specification.

`nscmd` – the same as `cmd`.

`nsecho` – the same as `echo`.

Message Identifier Matcher

Name: `/par/[port]/jps/N/id (N=[0…2])`
Access: `rw`
Type: `integer`
Values: `[-1…255]`
Default: `-1`

`-1` – matches any message.

`[0…255]` – matches a message which first byte is equal to the value of the field. For the [>>] message, its first byte is its `id` field. For non-standard message, the first byte is the message is its first character (lying between “!” and “/” in ASCII).

Skip the First Byte

Name: `/par/[port]/jps/N/skip (N=[0…2])`
Access: `rw`
Type: `boolean`
Values: `y,n`
Default: `y`

`y` – the first byte of the message is skipped before executing the message contents.

`n` – the first byte of the message is not skipped.

Source/Destination Port

Name: `/par/[port]/jps/N/port (N=[0…2])`
Access: `rw`
Type: `enumerated`
Values: `/[oport],(empty string)`
Default: `(empty string)`

If the value of the `mode` field is set to either `echo` or `nsecho` (see the parameter `/par/[port]/jps/N/mode` on page 220), this parameter specifies the output port to which the message contents should be sent.

For all other values of the `mode` field, this parameter specifies the input port so that the message contents is executed as if the data have been received through the specified port.

The default value, empty string, designates the current port, i.e., the port through which the initial message has been received.

Examples

Example: Suppose we have a controller connected to the serial A port of the receiver and we need to send both commands and CMR differential messages to the receiver using this port. Suppose also that the controller is capable to wrap CMR data into the [>>] messages and uses the `id` field of the [>>] message as part of CMR data; and we wish to pass these CMR data to the receiver as if they were received from serial C port. Suppose also that the controller is capable to send GREIS commands to the receiver prepending them by the '`!`' character. The receiver could be then configured using the following commands:

⇒ `set,/par/dev/ser/a/jps/0,{nscmd,33,y,""}`

⇒ `set,/par/dev/ser/a/jps/1,{cmr,-1,n,/dev/ser/c}`

⇒ `set,/par/dev/ser/a/jps/2/mode,none`

⇒ `set,/par/dev/ser/a/imode,jps`

⇒ `set,/par/dev/ser/c/imode,cmr`

Note: In the first command, 33 is decimal value of the ASCII code of the '`!`' character that our example controller will be prepending to GREIS commands after the port is configured as shown.

Note: Note that in the last command the port (`/dev/ser/c` in our example) should match those one specified in the second command as a virtual source of CMR data.

Example: Suppose that unlike previous example, the controller always sends GREIS commands prepended by command identifier (containing any string surrounded by '`%`' characters). Then we would instead wish to configure the 0-th specification as follows:

⇒ `set,/par/dev/ser/a/jps/0,{nscmd,37,n,""}`

where 37 is ASCII code of the '%' character, and we've changed the value of `skip` from `y` to `n`.

Example: Suppose that unlike previous examples, the controller sets `id` field of generated [>>] messages to decimal value 43, and puts CMR data only into the `data` field of the message. Then we will program the 1-th specification as follows:

⇒ `set,/par/dev/ser/a/jps/1,{cmr,43,y,/dev/ser/c}`

●

3.4.25 Serial Port Parameters

In this section, `[sport]` denotes a serial (RS232) port, – any of `/dev/ser/X (X=[a…d])`.

Hardware Settings

Baud Rate

Name: `/par/[sport]/rate`
Access: `rw`
Type: `integer`
Values: `460800,230400,153600,115200,57600,38400,19200,9600,4800,2400,1200,600,300`
Default: `&def`

The attribute `/par/[sport]/rate&def` specifies the default value for this parameter.

Name: `/par/[sport]/rate&def`
Access: `rw`
Type: `enumerated`
Values: `460800,230400,153600,115200,57600,38400,19200,9600,4800,2400,1200,600,300`
Default: `115200`

RTS/CTS Handshake

Name: `/par/[sport]/rtscts`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `off`

RTS State

Name: /par/[sport]/rts
Access: rw
Type: boolean
Values: on,off
Default: on

CTS State

Name: /par/[sport]/cts
Access: r
Type: boolean
Values: on,off

Number of Data Bits

Name: /par/[sport]/bits
Access: rw
Values: 5,6,7,8
Default: 8

Number of Stop Bits

Name: /par/[sport]/stops
Access: rw
Type: integer
Values: 1,2
Default: 1

Parity

Name: /par/[sport]/parity
Access: rw
Type: enumerated
Values: N,odd,even,fodd,feven
Default: N

N - no parity

odd - odd parity

even - even parity

fodd - forced odd parity (logical 1)

feven - forced even parity (logical 0)

Infrared Mode

Name: `/par/[sport]/ir`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `off`

Note that the JAVAD GNSS receiver may have either one infrared port, which is always port D, or no infrared port.

Output Time-frames

Overview

This feature allows to use the receiver in a time-sharing network where every receiver in a network is only allowed to send its data during specific time intervals.

Output time-frame is a periodic time interval of a specified length. Time-frames are entirely specified by three parameters: `period`, `length`, and `delay`. By definition, a time-frame begins the `delay` seconds after the receiver time modulo `period` becomes zero, and lasts for the `length` seconds.

While no time-frame is active, the data to be output to the port is buffered inside the receiver. As soon as next time-frame begins, receiver starts to output data to the port and keeps output allowed till the end of the time-frame. At the end of the time-frame receiver clears its internal buffers not to allow the data that reminded buffered to be output at the subsequent time frame.

Time-frame Mode

Name: `/par/[sport]/oframe/mode`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `off`

`on` – corresponding port will output data only at the specified time-frames.

`off` – data output to corresponding port will be enabled all the time.

Time-frame Period

Name: `/par/[sport]/oframe/period`
Access: `rw`
Type: `float [seconds]`
Values: `[0…86400]`
Default: `1`

This parameter specifies the frequency at which output time-frames will occur. Time-frames will be scheduled to be started at the moments when the receiver time modulo `period` is equal to zero. Note however that a time-frame is actually started the delay seconds later.

Time-frame Length

Name:	`/par/[sport]/oframe/length`
Access:	`rw`
Type:	`float [seconds]`
Values:	`[0…86400]`
Default:	`1`

This parameter specifies how long output will be enabled once a time-frame has been started.

Time-frame Delay

Name:	`/par/[sport]/oframe/delay`
Access:	`rw`
Type:	`float [seconds]`
Values:	`[0…86400]`
Default:	`0`

This parameter specifies how much time passes between the moment the output time-frame is scheduled, and when the time-frame actually begins.

3.4.26 Network Parameters

Note: Changes to some of these parameters may take effect only after receiver reboot.

LAN Configuration

These parameters aid in configuration of your receiver to be part of a TCP/IP local area network (LAN). Only static configuration defined by the parameters is currently supported (i.e., there is no support for DHCP). Note that all these parameters are sticky and are not reset to their default values when receiver NVRAM is cleared or parameters are initialized by the `init` command.

Receiver IP Address

Name: /par/net/ip/addr
Access: rw
Type: ip_address
Values: (any valid IP address)
Default: 192.168.2.2

This parameter identifies the receiver on a TCP/IP network.

Network mask

Name: /par/net/ip/mask
Access: rw
Type: ip_address
Values: (any valid IP address)
Default: 255.255.255.192

This parameter specifies the network mask of the local network the receiver is connected to.

Default Gateway

Name: /par/net/ip/gw
Access: rw
Type: string
Values: (any valid IP address)
Default: 192.168.2.1

The default gateway to use for packets that don't belong to the local network.

Maximum Transmission Unit (MTU)

Name: /par/net/ip/mtu
Access: rw
Type: integer
Values: [128…16384]
Default: 1500

The MTU for the interface.

MAC Address

Name: /par/net/mac/addr
Access: rw
Type: string
Values: (any valid MAC address)
Default: (automatically generated unique value)

This parameter specifies receiver's unique hardware number on a network. High part of the MAC address is fixed and is equal to `00:18:D7`. Low part is generated automatically from the receiver ID. The user usually does not need to change the MAC address.

WLAN (WiFi) Configuration

These parameters aid in configuration of your receiver to be part of a TCP/IP wireless local area network (WLAN). Only static configuration defined by the parameters is currently supported (i.e., there is no support for DHCP). Note that all these parameters are sticky and are not reset to their default values when receiver NVRAM is cleared or parameters are initialized by the `init` command.

Note: The only supported method of identifying of the access point to be used for connection is specifying its access point ID using `/par/net/wlan/ap/id` parameter. Support for identification using SSID will be implemented shortly.

Note: Currently WPA is unsupported.

WLAN Mode

Name: `/par/net/wlan/mode`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `on`

WLAN Connection State

Name: `/par/net/wlan/state`
Access: `r`
Type: `enumerated`
Values: `off,on,associated,error`
Default: `on`

WLAN Error

Name: `/par/net/wlan/error`
Access: `r`
Type: `string[0…128]`
Default: `"none"`

Human-readable error string describing the last error happened.

WLAN Access Point ID

Name: `/par/net/wlan/ap/id`
Access: `rw`
Type: `string`
Values: `(any valid MAC address)`
Default: `"00:00:00:00:00:00"`

WLAN Access Point RSSI

Name: `/par/net/wlan/ap/rssi`
Access: `r`
Type: `integer`
Values: `[0…255]`
Default: `0`

Access point Received Signal Strength Indication (RSSI). Indicates the amount of power present in a received radio signal.

WLAN Keys

Name: `/par/net/wlan/ap/keyN (N=[1…4])`
Access: `w`
Type: `string [0…32]`
Values: `(any string)`
Default: `""`

The `N`'th key string for the access point.

To increase security, these parameters are write-only and can't be read back.

WLAN Receiver IP Address

Name: `/par/net/wlan/ip/addr`
Access: `rw`
Type: `ip_address`
Values: `(any valid IP address)`
Default: `192.168.2.2`

This parameter identifies the receiver on a TCP/IP network.

WLAN Network mask

Name: `/par/net/wlan/ip/mask`
Access: `rw`
Type: `ip_address`
Values: `(any valid IP address)`
Default: `255.255.255.192`

This parameter specifies the network mask of the wireless network the receiver is connected to.

WLAN Default Gateway

Name: `/par/net/wlan/ip/gw`
Access: `rw`
Type: `string`
Values: `(any valid IP address)`
Default: `192.168.2.1`

The default gateway to use for packets that don't belong to the wireless network.

WLAN Maximum Transmission Unit (MTU)

Name: `/par/net/wlan/ip/mtu`
Access: `rw`
Type: `integer`
Values: `[128…16384]`
Default: `1500`

The MTU for the interface.

WLAN MAC Address

Name: `/par/net/wlan/mac/addr`
Access: `rw`
Type: `string`
Values: `(any valid MAC address)`
Default: `(automatically generated unique value)`

This parameter specifies receiver's unique hardware number on a network. High part of the MAC address is fixed and is equal to `00:18:D7`. Low part is generated automatically from the receiver ID. The user usually does not need to change the MAC address.

GPRS/DIALUP (PPP) Configuration

These parameters aid in establishing of either GPRS or dialup connection to a provider of Internet services using point-to-point protocol (PPP).

To create PPP link through internal or external modem connected to receiver serial port, the user should set the mode of corresponding modem port (`/par/dev/modem/X/mode`, `X=[a…d]`) to either `gprs` (for GSM modem) or `dialup` (for GSM or analog modem), depending on the kind of required connection (GPRS or dialup). Receiver scans modem mode parameters for each port from `a` to `d` in order and selects the first one with the `mode` equal to `gprs` or `dialup`. It then creates PPP link over the selected modem port.

Should the `mode` parameter of the modem port on which PPP link is active be set to a value that differs from the current setting, the PPP connection for this port will be terminated and the firmware will repeat search for modem mode equal to `gprs` or `dialup` among modem ports (starting from `dev/modem/a`).

Examples

Example: Establish GPRS link through GSM modem connected to receiver serial port C. The example assumes BEELINE cellular operator and PIN code `1234`.

⇒ `set,/par/net/ppp/gprs/pdp/apn,internet.beeline.ru`

⇒ `set,/par/net/ppp/gprs/pdp/id,1`

⇒ `set,/par/net/ppp/gprs/passwd,beeline`

⇒ `set,/par/net/ppp/gprs/user,beeline`

⇒ `set,/par/net/ppp/gprs/dial,"*99***1#"`

⇒ `set,/par/modem/c/type,gsm`

⇒ `set,/par/modem/c/pin,1234`

⇒ `set,/par/modem/c/mode,gprs`

Example: Establish dialup PPP link over PSTN modem connected to receiver serial port C.

⇒ `set,/par/net/ppp/dialup/dial,96007000`

⇒ `set,/par/net/ppp/dialup/user,mtd0633877@dlp`

⇒ `set,/par/net/ppp/dialup/passwd,abc`

⇒ `set,/par/modem/c/type,pstn`

⇒ `set,/par/modem/c/mode,dialup`

●

PPP Configuration Parameters

PPP Connection State

Name:	`/par/net/ppp/state`
Access:	`r`
Type:	`enumerated`
Values:	`off,connecting,connected,disconnecting`
Default:	`off`

`off` – PPP conection is inactive.

`connecting` – receiver tries to establish PPP connection.

`connected` – PPP connection is up and running.

`disconnecting` – receiver is disconnecting from PPP peer.

PPP Error

Name: `/par/net/ppp/error`
Access: `r`
Type: `string[0…256]`
Values: `"none", (arbitrary string)`
Default: `"none"`

This parameter contains PPP error message(s), or `"none"` if there are no errors.

PPP Baud Rate

Name: `/par/net/ppp/speed`
Access: `r`
Type: `enumerated`
Values: `9600,19200,38400,57600,115200`

This parameter contains baud rate at which receiver talks to the peer of PPP connection or to the modem.

PPP Set Default Route

Name: `/par/net/ppp/route`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `on`

`on` – receiver will add default route to the system routing table using the PPP peer as the default gateway. Note that if receiver is simultaneously connected to the LAN, this will switch routing from the LAN default gateway to the PPP peer.

`off` – receiver won't add the PPP peer as the default route to the system routing table, unless `/par/net/tcp/addr` is set to `0.0.0.0`.

PPP Debugging

Name: `/par/net/ppp/debug`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `off`

`on` – PPP connection debugging facilities are enabled. The debugging information will be output to the receiver serial Port A.

`off` – PPP connection debugging facilities are disabled.

Enable PAP Authenticaion

Name: `/par/net/ppp/auth/pap`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `on`

`on` – enable use of unencrypted password authentication protocol (PAP).

`off` – receiver will refuse to authenticate itself to the peer using PAP.

Enable CHAP Authentication

Name: `/par/net/ppp/auth/chap`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `on`

`on` – enable use of challenge handshake authentication protocol (CHAP).

`off` – receiver will refuse to authenticate itself to the peer using CHAP.

Enable Van Jacobson Compression

Name: `/par/net/ppp/comp/vj`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `off`

`on` – Van Jacobson style TCP/IP header compression (VJ) is enabled in both the transmit and receive directions.

`off` – the VJ compression is disabled.

Enable Connection-ID Compression

Name: `/par/net/ppp/comp/vjc`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `off`

`on` – receiver will omit the connection-ID byte from VJ-compressed headers and will ask the peer to do so.

`off` – receiver will not omit the connection-ID byte, nor ask the peer to do so.

GPRS Configuration

GPRS Dial Number

Name: `/par/net/ppp/gprs/dial`
Access: `rw`
Type: `string[0…32]`
Default: `"*99***1#"`

This parameter specifies the dial number for GPRS connection.

GPRS User Name

Name: `/par/net/ppp/gprs/user`
Access: `rw`
Type: `string[0…32]`
Default: `(empty string)`

This parameter specifies GPRS user name.

GPRS Password

Name: `/par/net/ppp/gprs/passwd`
Access: `rw`
Type: `string[0…32]`
Default: `(empty string)`

This parameter specifies GPRS password.

Note: This parameter is never printed implicitly.

GPRS PDP Context Identifier

Name: `/par/net/ppp/gprs/pdp/id`
Access: `rw`
Type: `integer`
Values: `[1…4]`
Default: `1`

This parameter specifies Packet Data Protocol (PDP) context identifier.

GPRS PDP Access Point Name

Name: `/par/net/ppp/gprs/pdp/apn`
Access: `rw`
Type: `string[0…32]`
Values: `(arbitrary string)`
Default: `(empty string)`

This parameter specifies Packet Data Protocol (PDP) access point name.

GPRS QoS Requested Precedence Class

Name: `/par/net/ppp/gprs/at/cgqreq/prcd`
Access: `rw`
Type: `integer`
Values: `[0…3]`
Default: `0`

This parameter specifies the precedence class for Quality of Service Profile Requested (for `AT+CGQREQ` command).

GPRS QoS Requested Delay Class

Name: `/par/net/ppp/gprs/at/cgqreq/delay`
Access: `rw`
Type: `integer`
Values: `[0…4]`
Default: `0`

This parameter specifies the delay class for Quality of Service Profile Requested (for `AT+CGQREQ` command).

GPRS QoS Requested Reliability Class

Name: `/par/net/ppp/gprs/at/cgqreq/relb`
Access: `rw`
Type: `integer`
Values: `[0…5]`
Default: `0`

This parameter specifies the reliability class for Quality of Service Profile Requested (for `AT+CGQREQ` command).

GPRS QoS Requested Peak Throughput Class

Name: `/par/net/ppp/gprs/at/cgqreq/peak`
Access: `rw`
Type: `integer`
Values: `[0…9]`
Default: `0`

This parameter specifies the peak throughput class for Quality of Service Profile Requested (for `AT+CGQREQ` command).

GPRS QoS Requested Mean Throughput Class

Name: `/par/net/ppp/gprs/at/cgqreq/mean`
Access: `rw`
Type: `integer`
Values: `[0…31]`
Default: `0`

This parameter specifies the mean throughput class for Quality of Service Profile Requested (for `AT+CGQREQ` command).

GPRS QoS Minimum Precedence Class

Name: `/par/net/ppp/gprs/at/cgqmin/prcd`
Access: `rw`
Type: `integer`
Values: `[0…3]`
Default: `0`

This parameter specifies the precedence class for Quality of Service Profile Minimum acceptable (for `AT+CGQMIN` command).

GPRS QoS Minimum Delay Class

Name: `/par/net/ppp/gprs/at/cgqmin/delay`
Access: `rw`
Type: `integer`
Values: `[0…4]`
Default: `0`

This parameter specifies the delay class for Quality of Service Profile Minimum acceptable (for `AT+CGQMIN` command).

GPRS QoS Minimum Reliability Class

Name: `/par/net/ppp/gprs/at/cgqmin/relb`
Access: `rw`
Type: `integer`
Values: `[0…5]`
Default: `0`

This parameter specifies the reliability class for Quality of Service Profile Minimum acceptable (for `AT+CGQMIN` command).

GPRS QoS Minimum Peak Throughput Class

Name:	`/par/net/ppp/gprs/at/cgqmin/peak`
Access:	`rw`
Type:	`integer`
Values:	`[0…9]`
Default:	`0`

This parameter specifies the peak throughput class for Quality of Service Profile Minimum acceptable (for `AT+CGQMIN` command).

GPRS QoS Minimum Mean Throughput Class

Name:	`/par/net/ppp/gprs/at/cgqmin/mean`
Access:	`rw`
Type:	`integer`
Values:	`[0…31]`
Default:	`0`

This parameter specifies the mean throughput class for Quality of Service Profile Minimum acceptable (for `AT+CGQMIN` command).

DIALUP Configuration

DIALUP Dial Number

Name:	`/par/net/ppp/dialup/dial`
Access:	`rw`
Type:	`string[0…32]`
Values:	`(arbitrary string)`
Default:	`(empty string)`

This parameter specifies dial number for dialup Internet provider.

DIALUP User Name

Name:	`/par/net/ppp/dialup/user`
Access:	`rw`
Type:	`string[0…32]`
Values:	`(arbitrary string)`
Default:	`(empty string)`

This parameter specifies user (login) name for dialup Internet provider.

DIALUP Password

Name: `/par/net/ppp/dialup/passwd`
Access: `rw`
Type: `string[0…32]`
Values: `(arbitrary string)`
Default: `(empty string)`

This parameter specifies password for dialup Internet provider.

Note: This parameter is never printed implicitly.

DIALUP Modem Initialization Script

Name: `/par/net/ppp/dialup/init`
Access: `rw`
Type: `string[0…64]`
Values: `(arbitrary string)`
Default: `"@AT@OK@ATI@OK@ATT@OK@"`

This parameter contains chat script to initialize dialup modem. The chat script defines a conversational exchange between the receiver and the modem. The syntax and semantics of the chat script used by the receiver matches those of the widely used "chat" program (see, e.g., *http://docs.freebsd.org/info/uucp/uucp.info.Chat_Scripts.html* for documentation), except the `'@'` character is used instead of cartridge return to separate chat commands.

Network Servers Parameters

Receiver implements FTP server (in read-only mode) and raw TCP server.

TCP/FTP Password

Name: `/par/net/passwd`
Access: `rw`
Type: `string[0…15]`
Values: `(arbitrary string)`
Default: `(automatically generated value unique for each receiver)`

By using this parameter the user sets a password for FTP and raw TCP connections.

Note: This parameter is never printed implicitly.

TCP Server Configuration

TCP Port

Name: `/par/net/tcp/port`
Access: `rw`
Type: `integer`
Values: `[1…65535]`
Default: `8002`

TCP port receiver is listening on for incoming raw TCP connections. Up to 5 simultaneous connections are supported. After a client connects to this TCP port, receiver outputs `login:` prompt. Explicit `a`, `b`,…, `e` reply followed by new line to the login prompt will make connection to corresponding TCP stream (`dev/tcp/a`,…,`dev/tcp/e`, respectively). Entering just new line will make receiver to automatically select the first available TCP stream to make connection to. After login prompt, the receiver will issue the password prompt. The string specified in `/par/net/passwd` followed by new line should be entered by the client in reply to the password prompt. If the string entered does not match, connection will be denied, otherwise connection to the selected raw TCP port will be established.

TCP Connection Timeout

Name: `/par/net/tcp/timeout`
Access: `rw`
Type: `integer [seconds]`
Values: `[1…0x7FFFFFFF]`
Default: `600`

The period of TCP connection inactivity after which the connection will be terminated.

FTP Server Configuration

FTP Port

Name: `/par/net/ftp/port`
Access: `rw`
Type: `integer`
Values: `[1…65535]`
Default: `21`

TCP port the receiver is listening on for incoming FTP connections. Only 1 simultaneous connection is supported.

FTP Connection Timeout

Name: `/par/net/ftp/timeout`
Access: `rw`
Type: `integer [seconds]`
Values: `[1… 0x7FFFFFFF]`
Default: `600`

The period of FTP connection inactivity after which the connection will be terminated.

UDP Parameters

Receiver supports data output using UDP packets. You will use receiver port name `dev/udp/a` with `em` or `out` commands to request UDP output. Parameters described in this section specify destination of UDP packets generated by the receiver as well as the rules of splitting of output data stream of bytes into stream of UDP packets.

UDP Destination Address

Name: `/par/dev/udp/a/addr`
Access: `rw`
Type: `string`
Values: `(any valid IP address)`
Default: `255.255.255.255`

This parameter specifies destination address for UDP packets. `255.255.255.255` is considered to be LAN broadcast address. Other kinds of broadcast and multicast addresses are also supported.

UDP Destination Port

Name: `/par/dev/udp/a/port`
Access: `rw`
Type: `integer`
Values: `[1…65535]`
Default: `8004`

This parameter specifies destination port for UDP packets.

Output Size Margin for UDP Packets

Name: `/par/dev/udp/a/omargin`
Access: `rw`
Type: `integer [bytes]`
Values: `[0…1200]`

This parameter specifies output size margin for UDP packets. As soon as number of bytes to be output exceeds the value of this parameter, UDP packet containing the data

will be sent. Note that receiver will never split single GREIS message between multiple UDP packets, so typical sizes of UDP packets will be greater than the value of this parameter. The end of epoch will cause sending of UDP packet provided the number of bytes remaining to be output is greater than zero, therefore the last packet in an epoch will be typically shorter than the value of this parameter.

TCP Client Parameters

Receiver is capable to operate as TCP client for different kinds of TCP servers. Connection to only one server at any given time is supported.

TCP Client Mode

Name:	`/par/net/tcpcl/mode`
Access:	`rw`
Type:	`enumerated`
Values:	`off,rcv,ntrip,jsrv`
Default:	`off`

`off` – TCP client is inactive.

`rcv` – use raw TCP connection to another receiver (RCV server).

`ntrip` – connect to NTRIP server.

`jsrv` – connect to JSRV server.

Note: JSRV server support is experimental and is intended for internal JAVAD GNSS use only.

When using `rcv` or `ntrip` mode, the TCP client port of the receiver, `dev/tcpcl/a`, should be configured to receive corresponding type of corrections but setting the port input mode accordingly. When using `jsrv` mode, the TCP client port should be set to the command input mode. Refer to “Input Mode” on page 214 for instructions on setting input mode of a port.

Note: We recommend to use `extrap` RTK mode (refer to “RTK Position Computation Mode” on page 122) due to potentially large delays on the Internet/GPRS .

TCP Client Connection State

Name:	`/par/net/tcpcl/state`
Access:	`r`
Type:	`enumerated`
Values:	`off,connecting,connected,disconnecting,error`

`off` – TCP client is inactive.

`connecting` – TCP client is connecting to the server.

`connected` – TCP client is connected to the server.

`disconnecting` – TCP client is disconnecting from the server.

`error` – TCP client was unable to connect to the server. In this case the parameter `/par/net/tcpcl/error` will contain the reason of the error.

TCP Client Error

Name: `/par/net/tcpcl/error`
Access: `r`
Type: `string[0…64]`
Values: `(arbitrary string)`
Default: `"none"`

This parameter will contain human-readable description of the reason of TCP client failure (if any).

RCV Mode Parameters

The parameters below are useful to provide a method to establish raw TCP connection to another (remote) JAVAD GNSS receiver, request data from the remote receiver, and then use the data as RTK/DGPS corrections.

Example: Configure receiver to connect to the reference receiver's raw TCP port B and receive RTCM corrections from the reference receiver:

```
⇒ set,/par/net/tcpcl/rcv/addr,172.17.0.34
⇒ set,/par/net/tcpcl/rcv/port,8002
⇒ set,/par/net/tcpcl/rcv/login,b        # login to TCP port B
⇒ set,/par/net/tcpcl/rcv/passwd,abc
⇒ set,/par/dev/tcpcl/a/imode,rtcm       # expect RTCM corrections
⇒ set,/par/net/tcpcl/mode,rcv
```

For this example to work, the reference receiver should be configured something like this:

```
⇒ set,/par/net/ip/addr,172.17.0.34
⇒ set,/par/net/ip/mask,255.255.255.0
⇒ set,/par/net/ip/gw,172.17.0.1
⇒ set,/par/net/tcp/port,8002
⇒ set,/par/net/passwd,abc
⇒ em,/dev/tcp/b,/msg/rtcm/{18,19,22,3}
⇒ set,/par/reset,y                    # for network changes to take effect
```

●

IP Address of Raw TCP Server

Name: `/par/net/tcpcl/rcv/addr`
Access: `rw`
Type: `string`
Values: `(any valid IP address)`
Default: `0.0.0.0`

The value of this parameter should match the IP address of the remote receiver.

IP Port of Raw TCP Server

Name: `/par/net/tcpcl/rcv/port`
Access: `rw`
Type: `integer`
Values: `[0…65535]`
Default: `0`

The value of this parameter should match the value of the parameter `/par/net/tcp/port` of the remote receiver.

Login Name for Raw TCP Server

Name: `/par/net/tcpcl/rcv/login`
Access: `rw`
Type: `enumerated`
Values: `a,b,c,d,e,(empty string)`
Default: `(empty string)`

This parameter specifies the login name to be sent to the remote receiver as the reply to the login prompt. Refer to the description of the `/par/net/tcp/port` parameter on page 239 for details.

Login Password for Raw TCP Server

Name: `/par/net/tcpcl/rcv/passwd`
Access: `rw`
Type: `string[0…32]`
Default: `(empty string)`

The value of this parameter should match the value of the parameter `/par/net/passwd` of the remote receiver.

Note: This parameter is never printed implicitly.

NTRIP Client Parameters

The parameters below are useful to provide a method to establish connection to an NTRIP caster, request data from particular mount point, and then receive and use the data as RTK/DGPS corrections.

Example: Configure receiver to connect to the NTRIP caster at specific IP address and port, to request data from the mountpoint `REF1` (that we know is sending RTCM corrections), and to receive RTCM corrections from the mount point. Suppose also that this NTRIP server requires NMEA GGA message to be sent to it periodically.

Note: You can obtain information about endpoints from the NTRIP table. To request NTRIP table, use `/par/net/tcpcl/ntrip/table` parameter (see below).

```
⇒ set,/par/net/tcpcl/ntrip/addr,87.236.81.134
⇒ set,/par/net/tcpcl/ntrip/port,80
⇒ set,/par/net/tcpcl/ntrip/mountpt,REF1
⇒ set,/par/net/tcpcl/ntrip/user,abc
⇒ set,/par/net/tcpcl/ntrip/passwd,abc
⇒ set,/par/net/tcpcl/ntrip/nmea,10          # send GGA every 10 seconds
⇒ set,/par/dev/tcpcl/a/imode,rtcm           # mountpoint sends RTCM
⇒ set,/par/net/tcpcl/mode,ntrip
```

●

IP Address of NTRIP Caster

Name: `/par/net/tcpcl/ntrip/addr`
Access: `rw`
Type: `string`
Values: `(any valid IP address)`
Default: `0.0.0.0`

The value of this parameter should match the IP address of the NTRIP caster to use.

IP Port of NTRIP Caster

Name: `/par/net/tcpcl/ntrip/port`
Access: `rw`
Type: `integer`
Values: `[0…65535]`
Default: `0`

The value of this parameter should match the IP port the NTRIP caster is listenning on for connections.

NTRIP Mount Point

Name: `/par/net/tcpcl/ntrip/mountpt`
Access: `rw`
Type: `string[0…15]`
Values: `(arbitrary string)`
Default: `(empty string)`

This parameter specifies the mount point of the NTRIP caster to get data from.

NTRIP User Name

Name: `/par/net/tcpcl/ntrip/user`
Access: `rw`
Type: `string[0…32]`
Values: `(arbitrary string)`
Default: `(empty string)`

This parameter specifies user ID for the protected space of the requested mount point. Only basic authentication scheme is supported. If empty, no user or password values will be sent to the NTRIP caster.

NTRIP Password

Name: `/par/net/tcpcl/ntrip/passwd`
Access: `rw`
Type: `string[0…32]`
Values: `(arbitrary string)`
Default: `(empty string)`

This parameter specifies the password for the protected space of the requested mount point. Only basic authentication scheme is supported.

Note: This parameter is never printed implicitly.

NMEA GGA Period for NTRIP

Name: `/par/net/tcpcl/ntrip/nmea`
Access: `rw`
Type: `integer`
Values: `[-1…86400], seconds`
Default: `0`

`-1` – receiver will not send NMEA GGA messages to NTRIP caster.

`0` – receiver will send NMEA GGA message to NTRIP caster only once after connection to the caster is established.

`[1…86400]` – receiver will send NMEA GGA messages to the NTRIP caster periodically, every specified number of seconds.

NTRIP Source Table

Name: /par/net/tcpcl/ntrip/table
Access: r
Type: string

Printing this parameter forces receiver to request NTRIP source table from the NTRIP caster and output the table in the reply. Every line of the NTRIP source table will be output in a separate [RE] message.

Example:
```
⇒ print,/par/net/tcpcl/ntrip/table
⇐ RE014 SOURCETABLE 200 OK
⇐ RE020 Server: NTRIP Caster 1.5.8/1.0
⇐ RE01A Content-Type: text/plain
⇐ RE017 Content-Length: 11366
⇐ RE002
⇐ RE056 CAS;www.euref-ip.net;2101;EUREF-IP;BKG;0;DEU;50.12;8.69;
  http://www.euref-ip.net/home
⇐ RE092 NET;EUREF;EUREF;B;N;http://www.epncb.oma.be/euref_IP;
  http://www.epncb.oma.be:80/stations/log/skl;
  http://igs.bkg.bund.de/index_ntrip_reg.htm;none
⇐ RE07A STR;ACOR0;Coruna;RTCM 2.3;1(1),3(60),16,18(1),19(1);2;GPS;
  EUREF;ESP;43.36;351.60;0;0;LEICA GRX1200PRO;none;B;N;3000;IGNE
⇐ RE091 STR;ALAC0;Alicante;RTCM
  2.1;1(1),3(10),16,18(1),19(1),22(10),23(10),24(10),59;2;
  GPS;EUREF;ESP;38.34;359.52;0;0;TRIMBLE NETRS;none;B;N;5000;IGNE
⇐ RE091 STR;ALME0;Almeria;RTCM 2.3;1(1),3(10),18(1),19(1),22(10),23(10),
  24(10),59(10);2;GPS;EUREF;ESP;36.85;357.54;
  0;0;TRIMBLE NETRS;none;B;N;4000;IGNE
⇐ […]
⇐ RE010 ENDSOURCETABLE
```

JSRV Client Parameters

Note: JSRV server support is experimental and is intended for internal JAVAD GNSS use only.

The parameters below are useful to provide a method to get into connection with Javad Server.

Example:
```
⇒ set,/par/net/tcpcl/jsrv/addr,172.17.0.49
⇒ set,/par/net/tcpcl/jsrv/port,8003
⇒ set,/par/dev/tcpcl/a/imode,cmd
⇒ set,/par/net/tcpcl/mode,jsrv
```

IP Address of Javad Server

Name: `/par/net/tcpcl/jsrv/addr`
Access: `rw`
Type: `string`
Values: `(any valid IP address)`
Default: `0.0.0.0`

The value of this parameter should match the IP address of the Javad Server to use.

IP Port of Javad Server

Name: `/par/net/tcpcl/jsrv/port`
Access: `rw`
Type: `integer`
Values: `[0…65535]`
Default: `8003`

The value of this parameter should match the IP port the Javad Server is listenning on for connections.

Network Statistics

Note: The parameters described below are mostly intended for the use by receiver firmware developers and are subject to change at any time.

TCP/IP Network Statistics

Name: `/par/net/stat`
Access: `r`
Type: `list {tcpd,mbuf,tcp,udp,icmp,if,drv,mem}`

`tcpd` – a list of active TCP connections.

`mbuf`, `tcp`, `udp`, `icmp`, `if` – internal statistics of the TCP/IP stack. The description of these fields exceeds the scope of this document[15].

`drv` – statistics from low-level Ethernet driver.

`mem` – memory usage statistics for network subsystem memory pool.

List of Active TCP Connections

Name: `/par/net/stat/tcpd`
Access: `r`
Type: `list`

15. Details can be found in the FreeBSD documentation.

For every active TCP connection this list contains an entry with a name that is a number in the range `[0…4]`.

Active TCP Connection

Name: `/par/net/stat/tcpd/N (N=[0…4])`
Access: `r`
Type: `list {ip,port,dev}`

`ip` – IP address of a peer of the selected TCP connection.

`port` – IP port of a peer of the selected TCP connection.

`dev` – TCP device allocated for the selected TCP connection.

TCP Peer IP Address

Name: `/par/net/stat/tcpd/N/ip (N=[0…4])`
Access: `r`
Type: `ip_address`

TCP Peer IP Port

Name: `/par/net/stat/tcpd/N/port (N=[0…4])`
Access: `r`
Type: `integer`
Values: `[0…65635]`

TCP Device Allocated for TCP Connection

Name: `/par/net/stat/tcpd/N/dev (N=[0…4])`
Access: `r`
Type: `string`
Values: `/dev/tcp/X (X=[a…e]), (empty string)`

An empty string indicates that the connection has been established but the user has not yet been logged in, i.e., receiver waits for login and/or password to be entered by the peer.

Ethernet Driver Statistics

Name: `/par/net/stat/drv`
Access: `r`
Type: `list {rxints,txints,errints,rxskips,state}`

`rxints` – number of receive interrupts.

`txints` – number of transmit interrupts.

`errints` – number of error interrupts.

`rxskips` – number of receive packets the driver lost.

`state` – driver state flags.

Receive Interrupts Count

Name: `/par/net/stat/drv/rxints`
Access: `r`
Type: `integer`

Transmit Interrupts Count

Name: `/par/net/stat/drv/txints`
Access: `r`
Type: `integer`

Error Interrupts Count

Name: `/par/net/stat/drv/errints`
Access: `r`
Type: `integer`

Lost Packets Count

Name: `/par/net/stat/drv/rxskips`
Access: `r`
Type: `integer`

Driver State Flags

Name: `/par/net/stat/drv/state`
Access: `r`
Type: `integer`

Driver state flags in hexadecimal representation as the logical OR product of the following flags:

`0x01` – waiting for receive event

`0x02` – waiting for transmit event

`0x04` – waiting for packet transmit acknowledge

`0x08` – waiting for transmitter ready

`0x10` – sleeping

Memory Usage for Network Subsystem

Name: `/par/net/stat/mem`
Access: `r`
Type: `list {sz,sf,min,bf,max,bu,su,ac,fc,ls,ms}`

Refer to the `/par/stat/mem` parameter on page 269 for the description of fields.

3.4.27 GSM Modem Parameters

Currently receivers support up to four GSM modems, called `a`, `b`, `c`, and `d`.

Note: In JAVAD GNSS receivers, the internal GSM modem is usually hardware-wise connected to Port C. As for external GSM radio modems, it is common practice to connect such modems to the receiver's Port D if available.

Modem Mode

Name: `/par/modem/X/mode (X=[a…d])`
Access: `rw`
Type: `enumerated`
Values: `off,master,slave`
Default: `off`

This parameter specifies the mode to be used by the GSM modem connected to the corresponding receiver port to communicate with the remote GSM modem at the other end of the radio link.

`off` – modem is off.

`master` – receiver will try to dial in to the remote (slave) modem using the number specified by the parameter `/par/modem/X/dial` to call.

`slave` – modem will wait for incoming calls from a master modem.

`gprs` – receiver will try to establish GPRS connection over GSM modem.

`dialup` – receiver will try to establish dialup connection over GSM or analog modem.

Note: In the current firmware implementation, the master mode should be used for the rover receiver only, and the slave mode should be used for the base receiver only.

PIN Code

Name: `/par/modem/X/pin (X=[a…d])`
Access: `rw`
Type: `string[4]`
Values: `four decimal digits`
Default: `0000`

This parameter specifies the SIM card’s PIN code for a GSM modem.

Dial Number

Name: `/par/modem/X/dial (X=[a…d])`
Access: `rw`
Type: `string[0…32]`
Values: `up to 32 decimal digits`
Default: `(empty string)`

This parameter specifies the phone number that the GSM modem will dial when in master mode.

Modem Control State

Name: `/par/modem/X/state (X=[a…d])`
Access: `r`
Type: `enumerated`
Values: `off,detect,detected,registration,gregistration,`
`ready,ring,dialling,connect,discon,err`

This parameter shows the current modem control state.

`off` - modem control is turned off.

`detect` - searching for a modem on the corresponding port.

`detected` - modem has been detected. Modem initialization is in progress.

`registration` - modem is being registered in the network.

`gregistration` - modem is being registered in the GPRS network.

`ready` - modem has been initialized and registered in the GSM network. If the modem is in slave mode, it is ready to receive an incoming call. If the modem is in master mode, it is ready to dial in to the slave modem.

`dialing` - modem is dialing the selected phone number as specified by the parameter `/par/modem/X/dial` (in master mode only).

`ring` - an incoming call is being received (in slave mode only).

`connect` - connection has been established.

`discon` - connection has been broken ("disconnecting").

`err` - fatal error has occurred. In this case the user will need to change the parameter `/par/modem/X/mode` to `off`, fix the problem and then retry setting the required modem mode. See the parameter `/par/modem/X/err` for what specifically might have caused the error.

Last Detected Modem Error

Name: `/par/modem/X/err (X=[a…d])`
Access: `r`
Type: `enumerated`
Values: `none,detect,init,pin,net,busy,no_carrier,no_answer,gnet,gservice,gdenial`
Default: `none`

This parameter shows the last of the errors identified by the modem driver provided the value of `/par/modem/X/state` is `err`.

`none` – no errors have been detected.

`detect` – cannot find a modem on the port.

`init` – an initialization error has occurred.

`pin` – wrong PIN code.

`net` – a network error has occurred.

`busy` – the phone number is busy. To rectify this temporary problem, just call again at a later time.

`no_carrier` – cannot detect the carrier signal. This temporary error can occur if the second modem (at the other end of the radio link) has not been initialized or if there are some problems with the GSM network. The given GSM modem will continue to dial in to the modem on the other side of the radio link until the carrier signal is detected.

`no_answer` – no hang up is detected after a fixed network time-out.

`gnet` – GPRS network error.

`gservice` – error attaching to GPRS service.

`gdenial` – GPRS registration denied.

Note: The modem control will attempt to automatically fix a detected error in case the parameter `/par/modem/X/state` takes a value other than `err` or `off`. No user intervention is needed unless the parameter `/par/modem/X/state` turns out equal to `err`, which means that the modem control has not been able to fix the problem on its own.

Receiver Port the Modem is Connected to

Name: `/par/modem/X/port (X=[a…d])`
Access: `r`
Type: `string`
Values: `/dev/ser/x`

This parameter is used to get the name of the serial port that the modem is connected to.

Data Wait Timeout

Name: `/par/modem/X/rcvtimeout (X=[a…d])`
Access: `rw`
Type: `integer [seconds]`
Values: `[0…1000]`
Default: `5`

If the receiver has not received any data from the modem for `rcvtimeout` seconds, the modem will be disconnected and then re-initialized. If the parameter is set to 0, such control will be disabled.

Service Word Repeat Period

Name: `/par/modem/X/sndtime (X=[a…d])`
Access: `rw`
Type: `integer [seconds]`
Values: `[0…1000]`
Default: `2`

This parameter, which specifies a time interval, is used to ensure reliable communication between the pair of modems (master - slave) and avoid unnecessary modem reinitialization.The transmit modem will send the service word to the receive modem every `sndtime` seconds. Note that the service word will not affect the differential corrections (RTCM or CMR messages) in any way. If the parameter is set to zero, the service word will not be used in data transmission.

Note: To ensure reliable and secure modem communication, the parameter `/par/modem/X/sndtime` must be larger then the period of transmitting differential corrections. Also, care should be taken that the time `/par/modem/X/rcvtimeout` is greater than the service word repeat period by 2 to 3 seconds.

Network Type

Name: `/par/modem/X/type (X=[a…d])`
Access: `rw`
Type: `enumerated`
Values: `gsm,pstn`
Default: `gsm`

This parameter specifies the type of the network to use.

`gsm` – GSM network (for GSM modem).

`pstn` – Public Switched Telephone Network (for analog modem).

Cellular Operator Name

Name: `/par/modem/X/inf/cell/oper (X=[a…d])`
Access: `r`
Type: `string[0…50]`
Default: `"unknown"`

GSM/GPRS/EDGE coverage

Name: `/par/modem/X/inf/cell/cover (X=[a…d])`
Access: `r`
Type: `string[0…50]`
Default: `"none"`

GSM Signal Quality.

Name: `/par/modem/X/inf/cell/sq (X=[a…d])`
Access: `r`
Type: `string[0…50]`
Default: `"unknown"`

Modem Vendor

Name: `/par/modem/X/inf/dev/vendor (X=[a…d])`
Access: `r`
Type: `string[0…50]`
Default: `"unknown"`

Modem Model

Name: `/par/modem/X/inf/dev/model (X=[a…d])`
Access: `r`
Type: `string[0…50]`
Default: `"unknown"`

Modem Revision

Name: `/par/modem/X/inf/dev/rev (X=[a…d])`
Access: `r`
Type: `string[0…50]`
Default: `"unknown"`

Modem Serial Number

Name: `/par/modem/X/inf/dev/sn (X=[a…d])`
Access: `r`
Type: `string[0…50]`
Default: `"unknown"`

3.4.28 Advanced Power Management

Primary Control Points

External Power Voltage

Name: `/par/pwr/ext`
Access: `r`
Type: `float [volts]`

Receiver Board Voltage

Name: `/par/pwr/board`
Access: `r`
Type: `float [volts]`

This voltage is measured directly on the board excluding the voltage drop across the "power switching circuitry".

Backup Battery Voltage

Name: `/par/pwr/backup`
Access: `r`
Type: `float [volts]`

External Antenna Control Points

External Antenna Voltage

Name: `/par/pwr/extant`
Access: `r`
Type: `float [volts]`

Provided `/par/ant/curinp` parameter's value is `ext`, this parameter contains the voltage of the antenna power supply.

External Antenna Current

Name: `/par/pwr/extantdc`
Access: `r`
Type: `float [milliamperes]`

Provided `/par/ant/curinp` parameter's value is `ext`, this parameter contains the DC the antenna draws from the antenna power supply.

Secondary Control Points

Digital Part 3 Volt Power

Name: `/par/pwr/d3v`
Access: `r`
Type: `float [volts]`

RF Part 3 Volt Power

Name: `/par/pwr/a3v`
Access: `r`
Type: `float [volts]`

Digital Part 5 Volt Power

Name: `/par/pwr/d5v`
Access: `r`
Type: `float [volts]`

RF Part 5 Volt Power

Name: `/par/pwr/a5v`
Access: `r`
Type: `float [volts]`

Power Source

Power Source Mode

Name: `/par/pwr/mode`
Access: `rw`
Values: `mix,ext,a,b,auto`
Default: `auto`

`mix` – receiver will automatically identify and start using the power source with the highest voltage. It is not recommended to run the receiver in this mode for a long time unless it is powered by a high capacity external source (note that in mix mode power consumption is about 10% higher than in other modes).

`ext` – receiver will be powered by the external power source,

`a` – receiver will be powered by the battery A.

`b` – receiver will be powered by the battery B.

`auto` – receiver will automatically select the most appropriate power source according to the following algorithm: if an external power source is detected, the receiver

will use this. Otherwise, the receiver will be powered by one of its batteries. If both batteries are mounted, the receiver will start with battery A.

Current Power Source

Name: `/par/pwr/curmode`
Access: `r`
Values: `mix,ext,a,b`

This parameter reports actual power source. Mostly useful when `/par/pwr/mode` is set to `auto`.

Batteries Status and Charging

Battery Charging Mode

Name: `/par/pwr/charge/bat`
Access: `rw`
Values: `off,a,b,auto`
Default: `auto`

This parameter instructs the receiver to charge corresponding battery (or batteries).

`off` – battery charging is turned off.

`a` – charging battery A.

`b` – charging battery B.

`auto` – receiver will automatically control the course of battery charging. Receiver automatically detects all of the batteries attached to the receiver (A, B, or both) and, if both are mounted, charges them one after the other (beginning with battery A). Once batteries are fully charged, the receiver stops charging.

Note: You can start battery charging only after the `/par/pwr/mode` parameter has been set to ext or auto.

Note: Some of receiver models must be turned on for the battery to receive a charge.

Warning: *If this parameter is set to either `a` or `b`, the receiver will keep charging corresponding battery. Care should be taken not to overcharge the battery.*

Currently Charging Battery

Name: `/par/pwr/charge/curbat`
Access: `r`
Values: `off,a,b`

The battery currently being charged. Mostly useful when `/par/pwr/charge/mode` is set to `auto`.

Battery Charging Speed

Name: `/par/pwr/charge/speed`
Access: `rw`
Values: `normal,fast`
Default: `normal`

This parameter toggles the battery charge speed between normal and high. The normal mode is preferable.

Note: When “auto” is used to charge the battery, the actual battery charge speed may differ from the value of this parameter.

Current Battery Charging Speed

Name: `/par/pwr/charge/curspeed`
Access: `r`
Values: `normal,fast`

Current battery charging speed. Mostly useful when `/par/pwer/charge/mode` is set to `auto`.

Battery Voltage

Name: `/par/pwr/bat/[a|b]`
Access: `r`
Type: `float [volts]`

The voltage of batteries “a” and “b”, respectively.

Charger Output Voltage

Name: `/par/pwr/charger`
Access: `r`
Type: `float [volts]`

Provided battery is being charged, this parameter contains the voltage of the receiver internal charger.

Charger Output DC

Name: `/par/pwr/chdc`
Access: `r`
Type: `float [milliamperes]`

Provided battery is being charged, this parameter contains the DC drawn from the receiver internal charger.

Converter Voltage

Name: `/par/pwr/conv`
Access: `r`
Type: `float [volts]`

3.4.29 TriPad Parameters

The following parameters allow the user to set/query the receiver configuration data responsible for the TriPad FN button's functionality.

Appending data to a specific file

Name: `/par/button/file`
Access: `rw`
Type: `string[20]`
Default: `(empty string)`

This parameter instructs the receiver to append new data to a specific existing file (unless the receiver finds no file with this name) when starting data recording via the FN button. This parameter can be set to a string comprising up to 20 valid characters. This string designates the name of the file you have selected for data appending.

If you have specified an empty name, the receiver will assign the current log-file an “automatically created name” every time you use TriPad to start data recording. (Note that this automatically created file name will depend on both the file creation time (month and day) and some additional “letter suffices”. The latter are used to avoid confusion between files created on the same day).

Alternatively, suppose you have specified a non-empty file name, say NAME. If there is no log-file with this name in the receiver memory, pushing the FN key will instruct the receiver to create a new file named /`log/NAME`. Otherwise, the receiver will be using the existing file /`log/NAME` for appending new data.

TriPad <FN> Button Action

Name: `/par/button/action`
Access: `rw`
Type: `enumerated`
Values: `led,dyn`
Default: `led`

This allows the user to toggle the <FN> button functionality between LED blink mode and toggling of receiver dynamic model.

`led` – <FN> button short click[16] will change the receiver's LED blink mode.

`dyn` – <FN> button click will toggle between the static and dynamic receiver modes, provided data recording is active.

Every time you change the receiver dynamic model through TriPad, the receiver will output an appropriate free form event to the current log-file.

When data recording is active, you can easily distinguish between static and dynamic visually. If the <REC> LED blinks green, the current mode is dynamic, if it blinks yellow, the mode is static.

Initial Dynamic Mode

Name: `/par/button/dyn`
Access: `rw`
Type: `enumerated`
Values: `static,dynamic`
Default: `static`

When `/par/button/action` is set to `dyn`, this parameter will specify the initial dynamic mode for all of the new files opened through TriPad.

Toggle Automatic File Rotation Mode (AFRM) via TriPad

Name: `/par/button/rot`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `off`

`off` – TriPad <FN> button will turn data logging on and off.

`on` – TriPad <FN> button will turn AFRM on and off.

Turn Data Recording On at Startup

Name: `/par/button/auto`
Access: `rw`
Type: `enumerated`
Values: `on,off,always`
Default: `off`

`on` – should a power failure occur in the course of data recording, the receiver will then automatically open a new file and resume data recording when power is on again. From a functional point of view, this is equivalent to pushing the <FN> button to start data logging once the receiver is powered on again.

16. Press the button and release it in less than one second.

`always` – this case is similar to the previous one except that the auto-start mechanism will be launched at receiver start time irrespective of whether the power failure occurred while data recording or not.

`off` – receiver will not resume data logging after power failure.

Note: Setting this parameter to either `on` or `always` will not make the receiver itself automatically start when power is restored after a power failure, though recent receivers will remember their on or off status, and therefore will turn on when power is restored, provided power failure occurred when they were turned on.

3.4.30 CAN Ports Parameters

In this section, [cport] denotes a CAN port - any of `/dev/can/X (X=[a,b])`.

CAN messages that receiver accepts are specified by the two parameters: `/par/[cport]/sid/in/first` and `/par/[cport]/sid/in/cnt`. To be accepted by the receiver, the CAN message Standard Identifiers (SIDs) of the input CAN messages must be in the range [`first`…`first`+`cnt`-1]. In addition, receiver will use the received SIDs to establish relative order of received the CAN messages.

CAN messages that receiver generates have programmable SIDs. The SID starts with the value specified by the `/par/[cport]/sid/out/first` parameter and is incremented by one for every CAN message being output until number of SIDs in the sequence exceeds the value of parameter `/par/[cport]/sid/out/cnt`. Then the SID is reset to its first value and the process continues. For example, if `first` is set to 0x710 and `cnt` is set to 3, the output CAN messages will have the following SIDs:

```
0x710,0x711,0x712,0x710,0x711,…
```

CAN Baud Rate

Name:	`/par/[cport]/rate`
Access:	`rw`
Type:	`integer [kbit/s]`
Values:	`1000, 500, 250, 125`
Default:	`125`

CAN bus baud rate in kilobits per second.

First SID for Input CAN Messages

Name: `/par/[cport]/sid/in/first`
Access: `rw`
Type: `integer`
Values: `[0x000…0x7FF]`
Default: `0x700`

The Number of SIDs for Input CAN Messages.

Name: `/par/[cport]/sid/in/cnt`
Access: `rw`
Type: `integer`
Values: `[1…8]`
Default: `8`

First SID for Output CAN Messages.

Name: `/par/[cport]/sid/out/first`
Name: `rw`
Type: `integer`
Values: `[0x000…0x7FF]`
Default: `0x700`

The Number of SIDs for Output CAN Messages.

Name: `/par/[cport]/sid/out/cnt`
Access: `rw`
Type: `integer`
Values: `[1…8]`
Default: `8`

3.4.31 IRIG Modulator Parameters

Enable IRIG Signal Output

Name: `/par/dev/irig/out`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `off`

IRIG Refernece Time

Name: `/par/dev/irig/time`
Access: `rw`
Type: `enumerated`
Values: `utcusno,gps`
Default: `utcusno`

IRIG Signal Offset

Name: `/par/dev/irig/offs`
Access: `rw`
Type: `integer [ns]`
Values: `[-500000…500000]`
Default: `0`

This parameter specifies IRIG signal offset in nanoseconds. Positive value will delay the signal with respect to the reference time.

IRIG Signal Amplitude

Name: `/par/dev/irig/ampl`
Access: `rw`
Type: `integer`
Values: `[0…255]`
Default: `170`

3.4.32 Messages

Message Groups

There are several message groups supported by the receiver. Each group comprises several related messages. Each entry in a group specifies default scheduling parameters for corresponding message. In addition to using message groups and message names in the `list` and `print` commands, individual message names could be used in the `em`, `out`, and `dm` commands.

Example: List the names of all the supported NMEA messages:

```
⇒ list,/msg/nmea
⇐ RE03F{GGA,GBS,GLL,GMP,GNS,GRS,GSA,GST,GSV,HDT,RMC,ROT,VTG,ZDA,P_ATT}
```

Example: Print default scheduling parameters of the GREIS [GA] message:

```
⇒ print,/msg/jps/GA
⇐ RE011{0.00,0.00,0,0x2}
```

Example: Enable output of the NMEA GGA message into the current terminal using default scheduling parameters:

```
⇒ em,/cur/term,/msg/nmea/GGA
```

●

GREIS Message NAME

Name: `/msg/jps/NAME`
Access: `r`
Type: `sched_params`

This parameter contains default scheduling parameters for GREIS message `NAME`. In general, message `NAME` matches the two-letter message identifier (see "Standard Messages" on page 280). However, for messages that have non-alphanumeric identifiers, the names used differ from their identifiers (see "Standard Predefined Messages" on page 283).

NMEA Message NAME

Name: `/msg/nmea/NAME`
Access: `r`
Type: `sched_params`

This parameter contains default scheduling parameters for NMEA message `NAME`. Standard NMEA messages are called after their three-letters identifiers (e.g., `GGA`). Proprietary NMEA messages are called by their three-letters identifiers and using `P_` prefix (e.g., `P_ATT`).

RTCM 2.x Message NAME

Name: `/msg/rtcm/NAME`
Access: `r`
Type: `sched_params`

This parameter contains default scheduling parameters for RTCM 2.x message `NAME`. RTCM messages are called after their decimal identifiers.

RTCM 3.x Message NAME

Name: `/msg/rtcm3/NAME`
Access: `r`
Type: `sched_params`

This parameter contains default scheduling parameters for RTCM 3.x message `NAME`. RTCM 3.x messages are called after their decimal identifiers.

CMR Message NAME

Name: `/msg/cmr/NAME`
Access: `r`
Type: `sched_params`

This parameter contains default scheduling parameters for CMR message `NAME`. CMR messages are called after their decimal identifiers.

Message Sets

The main purpose of supporting message sets is to provide ability to enable output of multiple messages and specify their scheduling parameters using single object name, the name of a message set.

Unlike message groups, message sets may contain “unrelated” messages, i.e., messages taken from different message groups. To avoid name clashes, the entries in the message sets have names comprising both message group name and message name inside its group.

Also unlike message groups, message sets are customizable. You may add and remove messages to/from message sets, and you may change scheduling parameters of the messages in the message sets. Note that the contents of a message set is only relevant at the moment of enabling the output of the message(s) from the message set, and has no impact on the currently enabled messages.

Currently 49 is the maximum allowed number of messages in each message set.

Below are some examples of using the message sets. Refer to description of corresponding commands for details and more examples.

Example: Remove GREIS [EL] message from the default set of messages:

⇒ `remove,/msg/def/jps/EL`

Example: Remove all the messages from the default set of messages:

⇒ `remove,/msg/`

Example: Add NMEA GGA message to the default set of messages:

⇒ `create,/msg/def/nmea/GGA`

Example: Change scheduling parameters for GREIS [SI] message in the default set of messages:

⇒ `set,/msg/def/jps/SI,{1,0,0,0x2}`

Example: Enable all the messages currently in the default set of messages to be output to the current terminal using scheduling parameters specified for the messages in the default set:

⇒ `em,/cur/term,/msg/def`

Example: Restore the default value for the default set of messages:

⇒ `init,/msg/def`

●

Default Set of Messages

Name: `/msg/def`
Access: `rw`
Type: `list {sched_params,…,sched_params}`
Default: `(receiver dependent)`

This parameter contains the default set of messages. The default value of this parameter is designed to contain GREIS messages suitable for wide range of post-processing software.

This message set is implicitly used when receiver log-files are created using TriPad or through AFRM.

Warning: *Remember that some of the message types are critical for JAVAD GNSS post-processing software to be able to import and process GREIS files correctly. Care should be taken when customizing the default set of messages.*

Minimum Set of GREIS Messages for RTK

Name: `/msg/rtk/jps/min`
Access: `rw`
Type: `list {sched_params,…,sched_params}`
Default: `{jps/RT,jps/SI,jps/rc,jps/cp,jps/2r,jps/2p,jps/BI,jps/ET}`

This parameter contains minimum required set of GREIS messages suitable for RTK. Note that the default value indicated above corresponds to a dual-frequency receiver.

Maximum Set of GREIS Messages for RTK

Name: `/msg/rtk/jps/max`
Access: `rw`
Type: `list {sched_params,…,sched_params}`
Default: `{jps/RT,jps/SI,jps/rc,jps/cp,jps/DC,jps/EC,`
`jps/2r,jps/2p,jps/D2,jps/E2,jps/BI,jps/ET}`

This parameter contains maximum set of GREIS messages suitable for RTK. Note that the default value indicated above corresponds to a dual-frequency receiver.

Message Output Lists

For every output port, there is corresponding message output list that holds the messages being enabled to be output to this port along with their current scheduling parameters.

Similar to message sets, message output lists may contain “unrelated” messages, i.e., messages taken from different message groups. To avoid name clashes, the entries in the message output lists have names comprising both message group name and message name inside its group.

The contents of these output lists are implicitly modified by the `em`, `out`, and `dm` commands.

Currently 49 is the maximum allowed number of messages in each message output list.

Message Output List for a Port

Name: `/par/out/[oport]`
Access: `r`
Type: `list {sched_params,…,sched_params}`
Default: `{}`

This parameter contains the list of messages enabled for output to the `[oport]` along with their scheduling parameters. Messages are output in the same order in which they appear in the message output list.

Number of Messages Enabled for Output to a Port

Name: `/par/out/[oport]&count`
Access: `r`
Type: `integer`
Values: `[0…49]`
Default: `0`

This parameter contains number of messages currently being enabled to be output to the `[oport]`.

3.4.33 Miscellaneous parameters

Processor’s Clock Frequency

Name: `/par/cpu/frq`
Access: `r`
Type: `integer [MHz]`

Returns the processor’s clock frequency.

Processor Load Statistics

Name: `/par/load`
Access: `r`
Type: `list`

This parameter contains statistics of processor load gathered since last request of the value of this parameter.

All the elements of the list but the last one have the format:

```
{NAME,load,min,max}
```

where:

`NAME` - thread name

`load` - computed processor load (in percents) associated with this thread

`min` - computed minimum time (in milliseconds) per thread execution cycle

`max` - computed maximum time (in milliseconds) per thread execution cycle

The last element of the list is of special interest to the user. It has the following format:

```
{load,soft_err,hard_err}
```

where:

`load` - average processor load in percents

`soft_err` - number of detected soft real-time errors

`hard_err` - number of detected hard real-time errors

The average processor load is supposed to be less than 90, and the number of detected hard real-time errors must be zero.

Enter Exception Mode on Errors

Name: `/par/except`
Access: `rw`
Type: `boolean`
Values: `on,off`
Default: `on`

`on` - receiver will enter "exception mode" if an unrecoverable error in program execution is detected.

`off` - receiver will reset itself and continue running when an unrecoverable error in program execution is detected. Here "reset" means a hardware reset similar to power cycle.

FLASH Memory Waitstates.

Name: `/par/rcv/flash/waitstates`
Access: `r`
Type: `integer`
Values: `[0…9]`

This parameter indicates how “fast” installed FLASH memory chip used to store firmware is. The smaller the returned value, the faster the flash memory. For most JAVAD GNSS receivers, this parameter is either 1 or 3.

Receiver Board Temperature

Name: `/par/dev/thermo/out`
Access: `r`
Type: `float [Celsius degrees]`

ASIC Frequency

Name: `/par/asic/curfrq`
Access: `r`
Type: `enumerated`
Values: `low,high`

Free Space in the Receiver’s NVRAM

Name: `/par/dev/nvm/a/free`
Access: `r`
Type: `integer [bytes]`

Time From the Receiver’s Battery-Backed RTC

Name: `/par/dev/rtc/time`
Access: `r`
Type: `{sec,min,hour,day,month,year}`

Memory Allocation Statistics

Name: `/par/stat/mem`
Access: `r`
Type: `list {sz,sf,min,bf,max,bu,su,ac,fc,ls,ms}`

This parameter describes memory allocation statistics for the main memory pool. This parameter is intended for the JAVAD GNSS’s firmware developers and is subject to change at any time.

`sz` – pool size in bytes

`sf` – free memory

`min` – minimum free memory

`bf` – number of free blocks

`max` – maximum number of free blocks

`bu` – number of blocks used

`su` – memory currently in use

`ac` – number of allocations

`fc` – number of de-allocations (frees)

`ls` – number of iterations in the longest block search

`ms` – mean number of block search iterations

3.4.34 Receiver Options

Options Overview

Among the many capabilities of your JAVAD GNSS receiver there is a special class of capabilities which are referred to as “receiver options”. By default, receiver options are disabled so you have to take special measures to activate them. It can be done by uploading an Option Authorization File (OAF) to the receiver using desktop receiver control software.

Each option is characterized by the following descriptors:

- Option name
- Purchased value
- Leased value
- Expiration date of leased value
- Current value

A receiver option can be “purchased”, “leased”, or “purchased” and “leased” at the same time. A leased option is characterized by the leased value and the expiration date.

A purchased option is characterized only by the purchased value because such an option has no expiration date. Since any option can be “purchased” and “leased” at the same time, we should take into account all of the “purchased” and “leased” descriptors as a whole. It is the numerical descriptor “current” that serves this purpose. This descriptor indicates the value currently effective for the given option.

Note that “current” will either coincide with the larger of the purchased and leased values or be set to -1.

- If “current” equals -1, this means that the corresponding receiver option is not supported by the firmware version you use.
- If “current” equals zero, the corresponding receiver option is disabled.
- If “current” equals a positive integer, the option is enabled.

Options Parameters

With the following read-only parameters, you can retrieve information about the receiver options.

Complete Information About the Receiver Options

Name: `/par/opts`
Access: `r`
Type: `list {cind,_GPS,_GLO,…}`

The first entry in this list is the `/par/opts/cind` parameter described above. The rest are entries for every existing JAVAD GNSS option. Refer to “Supported Options” on page 272 for the list of options.

Complete Information About the Option NAME

Name: `/par/opts/NAME`
Access: `r`
Type: `list {cur,purchased,leased,date}`

Current Value of the Option NAME

Name: `/par/opts/NAME/cur`
Access: `r`
Type: `integer`
Values: `[-1…511]`

This parameter contains currently effective option value. It will either be set according to the larger of the `/par/opts/NAME/purchased` and `/par/opts/NAME/leased` values or will be set to `-1`.

`-1` – the option `NAME` is not supported either by the firmware version you use, or by the receiver hardware.

`0` – the option `NAME` is disabled.

`[1…511]` – option `NAME` is enabled. Refer to “Supported Options” on page 272 for the meaning of particular values for given option.

Purchased Value of the Option NAME

Name: `/par/opts/NAME/purchased`
Access: `r`
Type: `integer`
Values: `[0…511]`

Leased Value of the Option NAME

Name: `/par/opts/NAME/leased`
Access: `r`
Type: `integer`
Values: `[0…511]`

Expiration Date of the Option NAME

Name: `/par/opts/NAME/date`
Access: `r`
Type: `string`
Default: `0`

This parameter contains either 0 if no leased value is loaded, or expiration date of the leased value in the format “`ddmmyy`”, where:

`dd` - decimal day of month [01…31]

`mm` - decimal month number [01…12]

`yy` - decimal year [00…99]

Supported Options

The following table describes currently supported receiver options:

Table 3-2. Receiver Options

Name	Description
_GPS	Allows use of GPS satellites
_GLO	Allows use of GLONASS satellites
_L1_	Allows CA/L1 measurements
_L2_	Allows P/L2 and P/L1 measurements
_POS	Specifies the maximum allowed position update rate for single point and code differential positioning in Hz.
_RAW	Specifies the maximum measurement update rate in Hz.
_MEM	Specifies maximum memory space for raw data files. Allows to store at least N bytes of data, where: N = _MEM (_MEM=[0…128]) N = 128 + (_MEM - 128) * 16 (_MEM=[129…511])
COOP	Enable common loops.

Table 3-2. Receiver Options

Name	Description
_PPS	Enable PPS signals. 1 - one of the available PPS signals, PPS A, is enabled; 2 - both PPS signals are enabled.
EVNT	Enable Event signals. 1 - one of the available Event signals, Event A, is enabled; 2 - both Event signals are enabled.
_AJM	Enable Jamming Suppressor. 1 - export (non-US) suppressor is enabled; 2 - US-laws compatible (weaker) suppressor is enabled.
_MPR	Enable Code and Carrier Phase Multipath Suppressors.
_FRI	Enable external frequency input.
_FRO	Enable 20 MHz stable frequency output.
RS_A	Maximum allowed baud rate for serial port A (in kilo-baud). If this option is not loaded, the current value of this option will be set to 115, thus enabling the port and limiting its speed by 115200 baud.
RS_B	Maximum allowed baud rate for serial port B (in kilo-baud)
RS_C	Maximum allowed baud rate for serial port C (in kilo-baud)
RS_D	Maximum allowed baud rate for serial port D (in kilo-baud)
INFR	Enable the infrared port.
_PAR	Enable the parallel port.
_FRH	Enable frequency hopping mode of the internal modem.
_DSS	Enable spread spectrum mode of the internal modem.
RAIM	Enable RAIM.
_DTM	Enable datums other than WGS84 and PE90.
MAGN	Enable magnetic declination.
_GEO	Enable geoid model.
_WPT	(unsupported)
WAAS	Enable WAAS/EGNOS satellites.
OMNI	(unsupported)

Table 3-2. Receiver Options

Name	Description
RTMO	Enable RTCM messages. This is a bit-field option. If bit#0 is set, RTCM messages relating to code differential are enabled. If bit#1 is set, RTCM messages relating to carrier phase differential are enabled. Other bits are reserved.
RTMI	Maximum number of ports that could be simultaneously set to the `rtcm` input mode. [0…5]
CMRO	Enable CMR messages. This is a bit-field option. If bit#0 is set, the whole range of CMR messages are enabled. Other bits are reserved.
CMRI	Maximum number of ports that could be simultaneously set to the `cmr` input mode. [0,1]
CDIF	Enable code differential positioning mode. [0,1]
PDIF	Maximum allowed RTK position update rate in Hz. [0…20]
_CPH	Enable true carrier phase. 0 - integral doppler is output instead of true carrier phase. In this case the option PDIF will not be fully available because only float solutions can be obtained when RTK using integral doppler for true carrier phase. 1- true carrier phase is output. Note: In JNS receivers other than JGG20, HE_GG, and JNS100, this option is always set to unity.
ETHR	Enable Ethernet. [0,1]
_TCP	Maximum number of simultaneous raw TCP connections. [0…5]
_FTP	Maximum number of simultaneous FTP connections. [0,1]
_USB	Enable USB device interface. [0,1]
OCTO	Enables heading and attitude modes. [0,1,2] 1 - heading mode is enabled 2 - both heading and attitude modes are enabled
_LIM	Always set to zero (intended for internal purposes only).
AUTH	Authorization for external programs. This is a bit-field option with the bits having the following definition [TBD]: bits #0…#7 – reserved.
JPSO	Enable GREIS messages. This is a bit-field option. If bit#0 is set, the whole range of GREIS messages are enabled. Other bits are reserved.
JPSI	Maximum number of ports that could be simultaneously set to the `jps` input mode. [0…5]

Table 3-2. Receiver Options

Name	Description
LAT1	Specifies the latitude of the upper left corner of the rectangle area within which the receiver can produce the position information and output measurement data. Measured in degrees from 0 to x90, where x = 0 stands for N – North hemisphere (positive numbers), x = 1 stands for S – South hemisphere (negative numbers)
LON1	Specifies the longitude of the upper left corner of the rectangle area within which the receiver can produce the position information and output measurement data. Measured in degrees from 0 to 360.
LAT2	Specifies the latitude of the lower right corner of the rectangle area within which the receiver can produce the position information and output measurement data. Measured in degrees from 0 to x90, where x = 0 stands for N – North hemisphere (positive numbers), x = 1 stands for S – South hemisphere (negative numbers)
LON2	Specifies the longitude of the upper left corner of the rectangle area within which the receiver can produce the position information and output measurement data. Measured in degrees from 0 to 360.

L_CS	Checksum of the LAT1, LON1, LAT2, LON2 options. This checksum is computed according to the following algorithm:
	`L_CS = LAT1 ^ LON1 ^ LAT2 ^ LON2`
	`if(L_CS == 0) L_CS = 1`
	If the checksum mismatches, then its current value is set to 0 and the receiver will not compute its position and output raw data measurements within corresponding rectangle.

LAT3	Specifies the latitude of the upper left corner of the second rectangle area within which the receiver can produce the position information and output measurement data. Measured in degrees from 0 to x90, where x = 0 stands for N – North hemisphere (positive numbers), x = 1 stands for S – South hemisphere (negative numbers)
LON3	Specifies the longitude of the upper left corner of the second rectangle area within which the receiver can produce the position information and output measurement data. Measured in degrees from 0 to 360.
LAT4	Specifies the latitude of the lower right corner of the second rectangle area within which the receiver can produce the position information and output measurement data. Measured in degrees from 0 to x90, where x = 0 stands for N – North hemisphere (positive numbers), x = 1 stands for S – South hemisphere (negative numbers)

Table 3-2. Receiver Options

Name	Description
LON4	Specifies the longitude of the upper left corner of the rectangle area within which the receiver can produce the position information and output measurement data. Measured in degrees from 0 to 360.

LCS2	Checksum of the LAT3, LON3, LAT4, LON4 options. This checksum is computed according to the following algorithm:
	`LCS2 = LAT3 ^ LON3 ^ LAT4 ^ LON4`
	`if(LCS2 == 0) LCS2 = 1`
	If the checksum mismatches, then its current value is set to 0 and the receiver will not compute its position and output raw data measurements within corresponding rectangle.

RM3I	Maximum number of ports that could be simultaneously set to the `rtcm3` input mode. [0…5]
RM3O	Enable RTCM3 messages. This is a bit-field option. If bit#0 is set, RTCM3 messages relating to code differential are enabled. If bit#1 is set, RTCM3 messages relating to carrier phase differential are enabled. Other bits are reserved.
_PPP	Enable point-to-point protocol (PPP) support.
TCCL	Enable TCP clients. This is a bit-field option. bit#0 - enable RCV client. bit#1 - enable NTRIP client. bit#2 - enable JSRV client.
UDPO	Enable messages output support over UDP/IP.

Receiver Messages

This chapter describes general format of GREIS standard messages as well as particular formats of all the predefined messages. Besides the GREIS standard messages, receiver supports quite a few messages of different formats, such as NMEA or BINEX. The formats of those "foreign" messages are described at the end of this chapter.

4.1 Conventions

4.1.1 Format Specifications

To describe some format as a sequence of bytes[1] in a compact form, we define formats for a few primary field types and then use notation close to those used in the C programming language to build definitions of more complex formats:

```
struct NAME {LENGTH} {
  TYPE FIELD[COUNT];  // DESCRIPTION
  …
  TYPE FIELD[COUNT];  // DESCRIPTION
};
```

where:

`NAME` – the name assigned to this format. It could be used in other format definitions as the `TYPE` of a field.

`LENGTH` – the length in bytes of entire sequence. For a fixed length format, it is a number, for a variable length message, it may be either an arithmetic expression depending on some other variable parameters or just the string `var`.

`TYPE FIELD[COUNT]` – field descriptor. It describes a sequence of `COUNT` elements of the same `TYPE` which is assigned the name `FIELD`. The `TYPE` could be either one of the primary field types described below, or a `NAME` of another format. When `[COUNT]` is absent, the field consists of exactly one element. When `COUNT` is absent (i.e., there are only empty square brackets, `[]`), it means that the field consists of unspecified number of elements.

1. In the context of this chapter, "byte" means 8-bit entity. Least significant bit of a byte has index zero.

`DESCRIPTION` – description of the field along with its measurement units and allowed range of values, where appropriate. Measurement units are surrounded by square brackets.

The following primary field types are defined:

Table 4-1. Primary Field Types

Type Name	Meaning	Length in Bytes
`a1`	ASCII character	1
`i1`	signed integer	1
`i2`	signed integer	2
`i4`	signed integer	4
`u1`	unsigned integer	1
`u2`	unsigned integer	2
`u4`	unsigned integer	4
`f4`	IEEE-754 single precision floating point	4
`f8`	IEEE-754 double precision floating point	8
`str`	zero-terminated sequence of ASCII characters	variable

To entirely define particular format, we also have to specify bytes order in the primary non-aggregate fields that are multi-byte (`i2`, `i4`, `u2`, `u4`, `f4`, `f8`). For GREIS messages this order is defined by the [MF] message, see “[MF] Messages Format” on page 288 for details.

Using the above definitions it’s possible to (recursively) expand any format specification to corresponding sequence of bytes. For example, the format

```
struct Example {9} {
  u1 n1;
  f4 n2;
  i2 n3[2];
};
```

expands to the following sequence of bytes assuming least significant byte first (LSB) order:

```
n1[0](0),
n2[0](0),n2[0](1),n2[0](2),n2[0](3),
n3[0](0),n3[0](1),n3[1](0),n3[1](1)
```

and to the following sequence of bytes assuming most significant byte first (MSB) order:

```
n1[0](0),
n2[0](3)n2[0](2)n2[0](1)n2[0](0)
n3[0](1)n3[0](0)n3[1](1)n3[1](0)
```

where `x[i](j)` designates `j`-th byte (byte #0 being least significant one) of an `i`-th element of the field `x`.

4.1.2 Special Values

For binary messages, some of their integer and floating point fields may contain special values, which are used instead of actual data when no data for the field are available. Binary fields for which checking for special values is required during data extraction are marked with the exclamation mark, “!” in the first column of the field definition.

The following table defines special values for various data field types:

Table 4-2. Special Values for Fields

Field Type	Special Value	HEX Representation
i1	127	7F
u1	255	FF
i2	32767	7FFF
u2	65535	FFFF
i4	2147483647	7FFF_FFFF
u4	4294967295	FFFF_FFFF
f4	quiet NaN	7FC0_0000
f8	quiet NaN	7FFF8_0000_0000_0000

4.2 Standard Message Stream

Standard GREIS message stream is a sequence of at most two kinds of messages, GREIS standard messages, and non-standard text messages.

Most important and widely used kind of messages is a rich set of GREIS standard messages. Their general format is carefully designed to allow for both binary and text mes-

sages, and to make it possible for applications to efficiently skip the messages the application doesn't know about or is not interested in.

Support for non-standard text messages, that should still adhere to the format defined for them in this manual, makes it possible to mix GREIS standard messages with messages of some other formats in the standard GREIS data stream. An example of such a format are NMEA messages.

Non-standard text messages of a special case, the messages that contain only ASCII <CR> and/or <LF> characters, are inserted by the message formatting engine in the receiver between the GREIS standard messages to make the resulting message stream more human-readable when it is sent to a terminal or generic text viewer or editor application.

Besides GREIS standard messages and non-standard text messages, JAVAD GNSS receivers typically support plenty of other formats (e.g., RTCM, BINEX, CMR). However, those formats are incompatible with the format of standard GREIS message stream. Should a stream contain messages of those formats, it can't be called GREIS standard message stream anymore, and can't be parsed by the same rules as the standard stream.[2]

4.3 General Format of Messages

4.3.1 Standard Messages

The format of every standard message is as follows:

```
struct StdMessage {var} {
  a1 id[2];          // Identifier
  a1 length[3];      // Hexadecimal body length, [000…FFF]
  u1 body[length];   // Body
};
```

Each standard message begins with the unique message identifier comprising two ASCII characters. Any characters from the subset "0" through "~" (i.e., decimal ASCII codes in the range of [48…126]) are allowed in identifier.

2. In fact, the format of GREIS standard messages is so flexible that it can incorporate any data stream into the standard GREIS data stream, but then the original incompatible stream should be wrapped into a sequence of special GREIS messages. The predefined message with identifier ">>" serves this purpose.

Message identifier is followed by the length of message body field. This field, which comprises three upper-case hexadecimal digits, specifies the length of the message body in bytes. Thus the maximum message body length is 4095 (0xFFF) bytes.

Message body follows immediately after the length field and contains exactly the number of bytes specified by the length field. There are no restrictions on the contents of the message body implied by the general format. The format of the message body in a message is implicitly defined by the message identifier. Formats of message bodies of all the predefined messages

4.3.2 Non-standard Text Messages

The format of non-standard text messages is as follows:

```
struct NonStdTextMessage {var} {
  a1 id;          // Identifier, [!…/]
  a1 body[];      // Body of arbitrary length, [0…)
  a1 eom;         // End of message (<CR> or <LF>)
};
```

Message identifier is any character in the range [!… /] (decimal ASCII codes in the range [33…47]). Message identifier is optional. If absent, the message body should have length zero (i.e., should be absent as well).

Message body is a sequence of ASCII characters except <CR> (decimal code 13) and <LF> (decimal code 10) characters. No limitation on the body length is imposed by the format.

The end of message marker is either <CR> or <LF> character.

Note that the format allows for non-standard messages comprising only CR or LF characters. This feature allows to make standard GREIS message streams look more human-readable when outputting data to a general-purpose terminal or viewing with generic text viewer or editor.

One of the non-standard text message identifiers, the character “$”, is already reserved as the identifier for standard NMEA messages. No other non-standard text messages should use the “$” as identifier.

4.3.3 Parsing Message Stream

In this section, you will find some hints and tips on how to write code intended to parse a GREIS receiver's message streams. Although we are not going to discuss this subject in detail in this reference manual, we'd like to emphasize here that the standard message

format will allow you to effectively process/parse nearly any GREIS message stream you may encounter in practice.

Synchronization

When parsing a message stream, you first need to find nearest message boundary. This is what is usually called “synchronization”. Message synchronization is carried out when parsing is started or when synchronization is lost due to an error in the data stream. In fact, to simplify the algorithm, you may consider that you are already synchronized when you start to parse the data stream. If it happens that it’s not indeed the case, the parsing error should occur. You then skip one character from the input stream and pretend you are synchronized again. Such approach effectively eliminates synchronization task as a separate part of the parsing algorithm.

Note: Due to the fact that the errors rate in a reasonably useful data stream should be rather low, the synchronization shouldn’t be a frequent task. In addition, the GREIS data stream typically consists of rather short messages, so the distance to the nearest message boundary is typically small. Taking into account these considerations, there is no requirement for synchronization algorithm to be very fast.

Skipping to the Next Message

Having the length in the general format of the standard GREIS messages allows you to easily ignore messages without knowing the format of their body. We indeed strongly recommend writing parsers so that they do skip unknown messages.

To go from the current message to the next one, take the following steps:

1. Assume the current message starts at position “N”. Determine the current message length (decode characters ## N+2, N+3, N+4). Assume the message length is equal to L. Skip the first L+5 characters starting from position “N”.
2. Skip all of <CR> and <LF> characters (if any).

Strictly speaking, we do not recommend that you use in your parsing code any a priori information about the sizes and the contents of the message bodies. If you respect this recommendation, you will not have trouble with the parsing program should some of the messages be changed.

Note: The rules and hints on parsing of message bodies of the standard predefined GREIS messages are discussed later in “Parsing Message Bodies” on page 283.

4.4 Standard Predefined Messages

In this section we will familiarize the reader with the predefined set of standard GREIS messages. When referring to a message with the identifier XX, we use the notation [XX]. While most messages are called by their message identifier in GREIS, some of them, specifically those that have non-alphanumeric identifiers, have names that are different. For such messages the notation [XX](NN) is used, where XX is message identifier, and NN is message name to be used in the GREIS commands. For example the message [~~](RT) has header "~~" and is called `/msg/jps/RT` in GREIS commands.

This section defines the formats of the bodies for all the standard predefined messages. Bear in mind that in a data stream every message has a standard header defined by the general format as well.

4.4.1 Parsing Message Bodies

Allowed Format Extensions

Formats of binary messages having fixed message size allow to add more data fields in the future. New fields are allowed to be inserted only at the end of message body just before the checksum field (if any). Such modifications to the message bodies are considered to be format extensions, not incompatible changes.

Though standard GREIS text messages aren't messages with fixed message size, new fields may still appear in these messages in the future. New fields can be either inserted at the end of an existing text message just before the checksum field, or immediately before any right-hand brace (`}`). For example, a message that is currently read as:

```
…1,{21,22},3,@CS
```

can be later extended to

```
…1,{2.1,2.2,2.3},3,4,@CS
```

where two additional fields, "2.3" and "4", were added.

Implement your parsing algorithms taking into account the following rules to make them work even with future format extensions:

1. Don't assume that the size of message body of the received message should exactly match specific size defined in this document. Only if the message is too short does it mean you can't use its contents. If the message is longer than expected, just ignore the excess data.
2. Address the checksum field relative to the end of the message body.

3. Address other data fields relative to the beginning of the message body.
4. Take into consideration the above rule for extending of text messages when writing data extractors for text messages.

Checksums

After a message has been extracted from the data stream using techniques described in the “Parsing Message Stream” on page 281, and the message identifier appears to be one of those the application is interested in, the message body should be parsed to extract the data. Before extracting the contents, the message checksum should be calculated and compared against the checksum contained in the message.

Most of predefined messages contain checksum. Checksum is computed using both the message header (i.e., “message identifier” plus “the length of message body”) and the body itself. See “Computing Checksums” on page 365 for more information on checksum computation.

The checksum is always put at the very end of the message body. If a message's structure is modified by adding a new data field(s), the new data fields will be added before the checksum field. This explains why it is recommended to address the checksum field relative to the end of the message body.

4.4.2 General Notes

Time Scales

There are five time scales your receiver may handle:

`Tr` – receiver time

`Tg` – GPS system time

`Tu` – UTC(USNO). Universal Coordinated Time supported by the U.S. Naval Observatory.

`Tn` – GLONASS system time.

`Ts` – UTC(SU). Universal Coordinated Time supported by the State Time and Frequency Service, Russia.

“Receiver time” is the only time grid that is always available in your receiver (i.e., the other time grids from the above list may or may not be currently available).

In fact, JAVAD GNSS receiver always synchronizes its local time (aka “receiver time”) with one of the four “global” time scales: GPS time, UTC(USNO), GLONASS time, or

UTC(SU). The time grid thus selected is referred to as “receiver reference time” (Trr) hereafter in this section[3].

Different time systems may have different time notations (formats) associated with them (e.g., for GPS time, we use such terms as “week number”, “time of week”, etc.). Note, however, that the “receiver time” representation will not depend on the selected receiver reference time and is always represented as receiver date and time of day.

Most of the predefined messages don't contain reference time information inside. In our view, it would be excessive to use one and the same time tag with all of the many messages the receiver generates at the current epoch. When outputting receiver information available for the current epoch, you usually get various messages. Instead of supplying each of them with an individual time tag data field, we use a special message that carries receiver time information common for these messages. This message is called “Receiver Time” and has the identifier [~~].

There is, however, a mode of operation, called RTK delayed mode, when at a given epoch receiver may produce solution referenced to some other epoch in the past. To provide time tag for such solution, special Solution Time-Tag [ST] message is used. In fact this message provides the correct time tag for a solution in all modes of operations, though in most modes it has exactly the same time as [~~].

There are some other messages having a time tag data field. Those are messages that contain information that appears independently on the receiver epoch grid. An example of such a message is “Event” [==].

Delimiters

In fact, “Receiver Time” message is supposed to precede all of the other messages generated at the current epoch thus delimiting messages corresponding to different epochs. From a formal point of view, it is up to the user to define the order of messages in the output stream. However, care should be taken to ensure that the order in which messages are written into the output stream does not break the “epoch synchronization”, which is very essential for post-processing the logged data with JAVAD GNSS software packages. For more details on the default set of messages see “Message Sets” on page 265.

For real-time applications it’s essential to determine the end of epoch as soon as possible. For such applications just delimiting epochs by a “start of epoch” marker is not convenient. We suggest to use the “Epoch Time” [::](ET) message as the “end of epoch” marker. This message contains the same time of day field that is found in the “Receiver Time” message that allows for better integrity checking. The idea is to compare time tag

3. In the current receiver firmware the receiver reference time is always GPS system time and can’t be customized.

from [::] message against the time tag from corresponding [~~] message. Mismatched tags are an indication of broken epoch.

You will notice that most of the messages have identifiers comprising only digits and/or English letters. In fact, “Receiver Time” [~~] is the only message whose identifier uses the character “~”. It makes sense as the [~~] message plays a very important part serving as an epoch delimiter. Thus there are special precautions in order to minimize the probability of losing this key message. Similarly, the identifier of the “Event” ([==]) message, too, must be as distinctive as possible since application software may use free-form events just as delimiters.

The idea of using “highly distinctive” identifiers for the messages that serve as delimiters is very clear. Should a message's checksum be wrong, just check its identifier. If neither of the identifier's characters coincides with “~”, then it is very unlikely that this is a corrupted [~~] message. Therefore, you needn't skip to the next [~~] message in this case.

On the other hand, if a message has the correct checksum yet one of the identifier characters is “~”, then it would be safer to treat this message as a corrupted [~~] message. In this case – skip to the next [~~] message.

Solution Types

The field “solType” used in many of the predefined messages designates the type of corresponding solution and may have the following values:

Table 4-3. Solution Types

Value	Meaning
1	stand-alone solution
2	code-differential (DGPS) solution
3	phase-differential (RTK) solution with float ambiguities
4	phase-differential (RTK) solution with fixed ambiguities
5	fixed. I.e., the value was entered, not calculated.

Satellite Navigation Status

Fields containing navigation status are used in a few of the predefined messages. Such fields designate the status of particular satellite with respect to position computation. The table below describes assigned values and their meanings.

Note: Codes 0…3 and 45…62 indicate that given satellite is used in position computation and show which measurements are used. The rest of codes indicate that satellite is not used in position computation and indicate why this satellite is excluded from position computation.

Table 4-4. Satellite Navigation Status

Value	Meaning
00	CA/L1 data used for position computation
01	P/L1 data used for position computation
02	P/L2 data used for position computation
03	Ionosphere-free combination used for position computation
04	Measurements are not available
05	Ephemeris is not available
06	Unhealthy SV (as follows from operational (=ephemeris) SV health)
07	Time-Frequency parameters from the ephemeris data set may be wrong[1]
08	Initial conditions (position and velocity vectors) from the ephemeris data set may be wrong[1]
09	Almanac SV health indicator is not available for this satellite[1]
10	Unhealthy SV (as follows from the almanac SV health indicator)[1]
11	“Alert” flag (from the word “HOW”) is set[2]
12	URA indicates the absence of accuracy prediction for this SV[2]
13	This SV is excluded from position computation by the user
14	SV with this frequency channel number is excluded from position computation by the user[1]
15	This SV is excluded from solution since its system number is unknown[1]
16	This SV has elevation lower than the specified mask angle
17	Reserved
18	Ephemeris data is too old
19	This SV does not belong to the constellation the user has selected
20	No data from reference station are available for given satellite (DGPS mode only)
21	Reserved
22	Wrong measurements have been detected by RAIM
23	SNR below specified minimum level
24	Reserved
25	Reserved
26	DLL is not settled
27	Ionospheric corrections are not received from base
28	Coarse code outlier has been detected
29	Reserved
30,31	No data from reference station are available for given satellite (RTK mode only)
32…50	Reserved
51	CA/L1 slot is used in RTK processing
52	P/L1 slot is used in RTK processing
53	P/L2 slot is used in RTK processing
54	P/L1 and P/L2 measurements are used in RTK processing
55	CA/L1 and P/L2 measurements are used in RTK processing
56…62	Reserved
63	Satellite navigation status is undefined

1. GLONASS only
2. GPS only

4.4.3 General Purpose Messages

[JP] File Identifier

```
struct FileId {85} {
  a1 id[5];            // File type identifier
  a1 description[80];  // Human-readable stream description
};
```

This message, that is intended to be put at the beginning of the file, serves two purposes. First, it enables the processing program to easily identify the file type. Second, this message usually contains some additional information about the origin of the corresponding file (e.g., what particular hardware was used to collect data this file contains).

Both the “id” and the “description” fields are padded to the required size with spaces if necessary.

For JAVAD GNSS receivers, the [JP] message always contains the following information: “id” = “RLOGF”, and “description” = “JPS NAME Receiver log-file” (blanks are omitted here), where the sub-string “NAME” stands for the specific receiver name.

Note: The size of this message is not subject to change. Therefore, the first 5 bytes of this message are always “JP055”, and specifically for this message generated by receiver, the first 10 bytes are always “JP055RLOGF”.

[MF] Messages Format

```
struct MsgFmt {9} {
  a1 id[2]="JP";     // JP identifier
  a1 majorVer[2];    // Format major version as decimal (e.g., '01')
  a1 minorVer[2];    // Format minor version as decimal
  a1 order;          // Bytes order
                     // '0' – LSB first;
                     // '1' – MSB first
  a1 crc[2];         // Checksum formatted as hexadecimal
};
```

Note: The size of this message is not subject to change. Therefore, the first 7 bytes of this message are always “MF009JP”.

The data field `order` describes how multi-byte binary types are stored inside the message bodies.

Note: For message format version 1.0, `order` is always set to “0”. Receiver always generates data in the least significant bytes first order.

The message format's major version is updated if and only if some backward incompatible changes to the existing message format are made. Any other changes to the existing messages result in updating only the minor version.

4.4.4 Time Messages

[~~](RT) Receiver Time[4]

This message contains the “time of day” part of the full receiver time representation (Tr).

```
struct RcvTime {5} {
  u4 tod;  // Tr modulo 1 day (86400000 ms) [ms]
  u1 cs;   // Checksum
};
```

This message is intended to be used as a “start of epoch” marker.

[::](ET) Epoch Time[5]

```
struct EpochTime {5} {
  u4 tod;  // Tr modulo 1 day (86400000 ms) [ms]
  u1 cs;   // Checksum
};
```

This message is intended to be used as an “end of epoch” marker. Provided the [~~](RT) message is used as “start of epoch” marker and [::](ET) is used as “end of epoch” marker, one can check that time tags from the messages from given epoch match to increase integrity checking capability of the stream decoding algorithm.

[RD] Receiver Date

```
struct RcvDate {6} {
  u2 year;   // Current year [1…65534][]
  u1 month;  // Current month [1…12] []
  u1 day;    // Current day [1…31] []
  u1 base;   // Receiver reference time [enumerated]
             //       0 – GPS
             //       1 – UTC_USNO
             //       2 – GLONASS
             //       3 – UTC_SU
             //   4…254 – Reserved
  u1 cs;     // Checksum
};
```

This message contains the “date” part of the full receiver time representation (Tr).

4. Use message name /msg/jps/RT to enable/disable the message.
5. Use message name /msg/jps/ET to enable/disable the message.

[GT] GPS Time

```
struct GPSTime {7} {
  u4 tow;  // Time of week [ms]
  u2 wn;   // GPS week number (modulo 1024) []
  u1 cs;   // Checksum
};
```

[TO] Reference Time to Receiver Time Offset

```
struct RcvTimeOffset {9} {
  f8 val;  // Trr - Tr [s]
  u1 cs;   // Checksum
};
```

[DO] Derivative of Receiver Time Offset

```
struct RcvTimeOffsetDot {5} {
  f4 val;  // Derivative of (Trr - Tr) [s/s]
  u1 cs;   // Checksum
};
```

[BP] Rough Accuracy of Time Approximation

```
struct RcvTimeAccuracy {5} {
  f4 acc;  // Accuracy [s]
  u1 cs;   // Checksum
};
```

If the value of accuracy is greater than 10^{-3}[s], it means that receiver clock may not be properly synchronized to receiver reference time (Trr).

[NT] GLONASS Time

```
struct GLOTime {7} {
  u4 tod;  // time of day [ms]
  u2 dn;   // GLONASS day number (modulo 4 years
           // starting from 1996) []
  u1 cs;   // Checksum
};
```

[NO] GLONASS to Receiver Time Offset

```
struct RcvGLOTimeOffset {6} {
  i1 sec;  // (Tn - Tr) [s]
  i4 ns;   // (Tn - Tr) modulo 1 second [ns]
  u1 cs;   // Checksum
};
```

[GO] GPS to Receiver Time Offset

```
struct RcvGPSTimeOffset {6} {
  i1 sec;  // (Tg - Tr) [s]
  i4 ns;   // (Tg - Tr) modulo 1 second [ns]
  u1 cs;   // Checksum
};
```

[UO] GPS UTC Time Parameters

```
struct GpsUtcParam {24} {
  UtcOffs utc; // GPS UTC time offset parameters
  u1 cs;       // Checksum
};

struct UtcOffs {23} {
  f8 a0;     // Constant term of polynomial [s]
  f4 a1;     // First order term of polynomial [s/s]
  u4 tot;    // Reference time of week [s]
  u2 wnt;    // Reference week number []
  i1 dtls;   // Delta time due to leap seconds [s]
  u1 dn;     // 'Future' reference day number [1…7] []
  u2 wnlsf;  // 'Future' reference week number []
  i1 dtlsf;  // 'Future' delta time due to leap seconds [s]
};
```

This message describes the relationship between UTC(USNO) and GPS time as specified by GPS subframe 4, page 18.

For how to convert GPS time into UTC(USNO), see ICD-GPS-200C, Revision IRN-200C-004 April 12, 2000.

[WU] WAAS UTC Time Parameters

```
struct WaasUtcParam {31} {
  UtcOffs utc; // WAAS to UTC time offset parameters
  i1 utcsi;    // UTC Standard Identifier[]
  u4 tow;      // Reference time of week [s]
  u2 wn;       // Reference week number []
  u1 cs;       // Checksum
};
```

This message has much in common with the [UO] message. The `utcsi` field may have one of the following values:

Table 4-5. UTC Standard Identifier

Value	Meaning
0	UTC as operated by the Communications Research Laboratory (CRL), Tokyo, Japan
1	UTC as operated by the National Institute of Standards and Technology (NIST)
2	UTC as operated by the U. S. Naval Observatory (USNO)

Table 4-5. UTC Standard Identifier

Value	Meaning
3	UTC as operated by the International Bureau of Weights and Measures (BIPM)
[4…7]	Reserved

[EU] GALILEO UTC and GPS Time Parameters

```
struct GalUtcGpsParam {xxx} {
  UtcOffs utc; // GALILEO to UTC time offset parameters
  // GALILEO to GPS time offset parameters
  f4 a0g;      // Constant term of time offset [s]
  f4 a1g;      // Rate of time offset [s/s]
  u4 t0g;      // Reference time of week
  u2 wn0g;     // Reference week number
  u1 cs;       // Checksum
};
```

4.4.5 Position/Velocity Messages

[ST] Solution Time-Tag

```
struct SolutionTime {6} {
  u4 time;     // Solution time. Tr modulo 1 day (86400000 ms)[ms]
  u1 solType;  // Solution type
  u1 cs;       // Checksum
};
```

Specifies the receiver time of the current position solution. Note that this time-tag may differ from the current receiver time if the receiver runs in RTK delay mode. In this case the time tag from this message is typically in the past with respect to the time tag of the current epoch.

[PO] Cartesian Position

```
struct Pos {30} {
  f8 x, y, z;  // Cartesian coordinates [m]
  f4 sigma;    // Position SEP6 [m]
  u1 solType;  // Solution type
  u1 cs;       // Checksum
};
```

6. SEP stands for *Spherical Error Probable*

[VE] Cartesian Velocity

```
struct Vel {18} {
  f4 x, y, z;  // Cartesian velocity vector [m/s]
  f4 sigma;    // Velocity SEP [m/s]
  u1 solType;  // Solution type
  u1 cs;       // Checksum
};
```

[PV] Cartesian Position and Velocity

```
struct PosVel {46} {
! f8 x, y, z;     // Cartesian coordinates [m]
! f4 pSigma;      // Position SEP [m]
! f4 vx, vy, vz;  // Cartesian velocities [m/s]
! f4 vSigma;      // Velocity SEP [m/s]
  u1 solType;     // Solution type
  u1 cs;          // Checksum
};
```

[PG] Geodetic Position

```
struct GeoPos {30} {
  f8 lat;      // Latitude [rad]
  f8 lon;      // Longitude [rad]
  f8 alt;      // Ellipsoidal height [m]
  f4 pSigma;   // Position SEP [m]
  u1 solType;  // Solution type
  u1 cs;       // Checksum
};
```

[VG] Geodetic Velocity

```
struct GeoVel {18} {
  f4 lat;      // Northing velocity [m/s]
  f4 lon;      // Easting velocity [m/s]
  f4 alt;      // Height velocity [m/s]
  f4 pSigma;   // Velocity SEP [m/s]
  u1 solType;  // Solution type
  u1 cs;       // Checksum
};
```

[SG] Position and Velocity RMS Errors

```
struct Rms {18} {
! f4 hpos;     // Horizontal position RMS error[m]
! f4 vpos;     // Vertical position RMS error [m]
! f4 hvel;     // Horizontal velocity RMS error [m/s]
! f4 vvel;     // Vertical velocity RMS error [m/s]
  u1 solType;  // Solution type
  u1 cs;       // Checksum
};
```

[DP] Dilution of Precision (DOP)

```
struct Dops {14} {
  f4 hdop;      // Horizontal dilution of precision (HDOP)[]
  f4 vdop;      // Vertical dilution of precision (VDOP) []
  f4 tdop;      // Time dilution of precision (TDOP) []
  u1 solType;   // Solution type
  u1 cs;        // Checksum
};
```

[SP] Position Covariance Matrix

```
struct PosCov {42} {
  f4 xx;        // [m^2]
  f4 yy;        // [m^2]
  f4 zz;        // [m^2]
  f4 tt;        // [m^2]
  f4 xy;        // [m^2]
  f4 xz;        // [m^2]
  f4 xt;        // [m^2]
  f4 yz;        // [m^2]
  f4 yt;        // [m^2]
  f4 zt;        // [m^2]
  u1 solType;   // Solution type
  u1 cs;        // Checksum
};
```

[SV] Velocity Covariance Matrix

```
struct VelCov {42} {
  f4 xx;        // [(m/s)^2]
  f4 yy;        // [(m/s)^2]
  f4 zz;        // [(m/s)^2]
  f4 tt;        // [(m/s)^2]
  f4 xy;        // [(m/s)^2]
  f4 xz;        // [(m/s)^2]
  f4 xt;        // [(m/s)^2]
  f4 yz;        // [(m/s)^2]
  f4 yt;        // [(m/s)^2]
  f4 zt;        // [(m/s)^2]
  u1 solType;   // Solution type
  u1 cs;        // Checksum
};
```

[BL] Base Line

```
struct BaseLine {34} {
  f8 x, y, z;   // Calculated baseline vector coordinates [m]
  f4 sigma;     // Baseline Spherical Error Probable (SEP) [m]
  u1 solType;   // Solution type
  i4 time;      // receiver time of the baseline estimate [s]
  u1 cs;        // Checksum
};
```

[PS] Position Statistics

```
struct PosStat {9} {
  u1 solType;    // Solution type
  u1 gpsLocked;  // Number of GPS SVs locked
  u1 gloLocked;  // Number of GLONASS SVs locked
  u1 gpsAvail;   // Number of GPS SVs available for positioning
  u1 gloAvail;   // Number of GLONASS SVs available for positioning
  u1 gpsUsed;    // Number of GPS SVs used in positioning
  u1 gloUsed;    // Number of GLONASS SVs used in positioning
  u1 fixProg;    // Ambiguity fixing progress indicator
                 // controllable by RTK engine [%]
  u1 cs;         // Checksum
};
```

The `fixProg` field may vary from 0% to 100%, though in practice if raw measurements are good enough, the `fixProg` field rarely takes values other than zero. Just occasionally you can see `fixProg` to be 100%. This means that the engine has just finished fixing all available ambiguities. The `fixProg` will be dropped to zero immediately after it has reached 100%.

If the `fixProg` field keeps varying between 0% and 100% exclusive, this means that not all of the ambiguities have been fixed. Here are possible reasons for such behavior:

- You have just launched the RTK engine and it is trying to get a first fixed solution.
- There is one or more problem satellites whose measurements prevent the engine from fixing all available ambiguities “in batch”.
- The receiver has just started tracking one or more “new” satellites. It will take the RTK engine some time to fix these new ambiguities.

Also note that if the solution type is “RTK fixed”, the number of SVs with float ambiguities is:

gpsAvail + gloAvail - (gpsUsed + gloUsed)

[PT] Time of Continuous Position Computation

```
struct PosCompTime {5} {
  u4 pt;  // Continuous position computation time [s]
  u1 cs;  // Checksum
};
```

Specifies the time interval over which continuous position computation has been possible. If the receiver is unable to compute any position at the current epoch, the Time of Continuous Position Computation counter is zeroed.

4.4.6 Satellite Measurements

In this section we will focus on messages containing "satellite specific information". These kinds of messages include satellite measurements (code and carrier phase measurements, elevations, azimuths, etc.).

Different applications may utilize different sets of measurements. It is almost impossible to select a fixed set of combinations of satellite measurements that would be enough universal yet compact. Instead receiver provides dedicated message for each particular measurement type. Every individual measurement message contains some specific ("homogeneous") data for the satellites tracked.

For any given epoch, data for particular satellite is put at the same position (index) in all the messages. Special "Satellite Indices" [SI] message establishes the mapping between Universal Satellite Identifier (USI) and the index for this particular satellite. The number of satellites an epoch has data for, `nSats`, should also be obtained from the [SI] message. This approach allows the minimization of the data transfer overheads (note that the gain increases as the number of satellites grows).

For multi-antenna receivers, given USI typically appears multiple times in every [SI] message, indexing observables obtained from different antennas. The [AN] message should then be used to determine which antenna the observables are obtained from.

Note: Most of the measurements messages may contain special values of corresponding types to indicate lack of data for particular satellite(s). Refer to Table 4-2, "Special Values for Fields," on page 279 for details.

Universal Satellite Identifier (USI)

To make it possible to handle data corresponding to satellites of different systems in a universal manner, we assign each satellite its Universal Satellite Identifier (USI).

The following table describes USI allocation:

Table 4-6. Universal Satellite Identifiers (USI) Allocation

USI Range	Assigned Satellites
0	Unused. Ignore satellites with this USI.
[1…37]	GPS PRNs [1…37]
[38…69]	GLONASS FCNs [-7…24]
70	GLONASS satellite with unknown FCN[1].
[71…119]	GALILEO [1…49]
[120…138]	WAAS PRNs [120…138]
[139…254]	Reserved

Table 4-6. Universal Satellite Identifiers (USI) Allocation

USI Range	Assigned Satellites
255	Unused. Ignore satellites with this USI.

1. Could be useful when converting third-party GLONASS measurement file into GREIS format.

[SI] Satellite Indices

```
struct SatIndex {nSats+1} {
  u1 usi[nSats];  // USI array []
  u1 cs;          // Checksum
};
```

The [SI] message contains an array of USIs indicating the currently tracked satellites that are above elevation angle selected for given output stream[7]. The number of satellites in the [SI] message, `nSats`, should be calculated as the length of message body (taken from the message header) minus one.

The [SI] message serves as a "reference map" for other measurement message types since it establishes the one-to-one correspondence between the satellite identifier and the array index allocated to this satellite. Bear in mind that this correspondence holds true until the next [SI] message is issued (note that the new [SI] message may or may not differ from the previous one).

[AN] Antenna Names

```
struct AntName{nSats+1} {
  a1 name[nSats];  // Antenna names[a…z]
  u1 cs;           // Checksum
};
```

This message contains antenna name (ASCII character in the range [a…z]) for every satellite from the latest [SI] message.

This message is only available for muli-antenna receivers.

[NN] GLONASS Satellite System Numbers

```
struct SatNumbers {nGloSats+1} {
! u1 osn[nGloSats];  // GLONASS SV orbit slot number []
  u1 cs;             // Checksum
};
```

The [NN] message contains the orbit slot number for every GLONASS satellite from the latest [SI] message. Here nGloSats designates the number of GLONASS satellites in the

7. As specified by /par/out/elm/[port] parameter.

corresponding [SI] message (remember that GLONASS satellites have USI lying within the range [38…70]).

[EL] Satellite Elevations

```
struct SatElevation {nSats+1} {
! i1 elev[nSats];  // Elevation angle [degrees] [-90…90)
  u1 cs;           // Checksum
};
```

This message contains elevations for all the satellites specified in the latest [SI] message.

[AZ] Satellite Azimuths

```
struct SatAzimuth {nSats+1} {
! u1 azim[nSats];  // Azimuth angle [degrees*2] [0…180)
  u1 cs;           // Checksum
};
```

This message contains azimuths for all the satellites specified in the latest [SI] message. The notation [degrees*2] means that the values from the message must be multiplied by 2 to restore actual azimuths in degrees.

[RC], [R1], [R2], [R3], [R5]: Pseudoranges

```
struct PR {8*nSats+1} {
! f8 pr[nSats];  // Pseudorange, [s]
  u1 cs;         // Checksum
};
```

These messages contain corresponding pseudoranges for all the satellites specified in the latest [SI] message. The [RC], [R1], [R2], [R3], and [R5] messages contain CA/L1, P/L1, P/L2, CA/L2, and L5 pseudoranges, respectively.

[rc], [r1], [r2], [r3], [r5]: Short Pseudoranges

```
struct SPR {4*nSats+1} {
! i4 spr[nSats];  // (PR[s] - 0.075) * 10^11
  u1 cs;          // Checksum
};
```

These messages contain corresponding short pseudoranges for all the satellites specified in the latest [SI] message. The [rc], [r1], [r2], [r3], and [r5] messages contain short CA/L1, P/L1, P/L2, CA/L2, and L5 pseudoranges, respectively.

Use the following formula to restore true pseudoranges in seconds:

$$pr = spr * 10^{-11} + 0.075$$

[1R], [2R], [3R], [5R]: Relative Pseudoranges

```
struct RPR {4*nSats+1} {
! f4 rpr[nSats];  // PR - CA/L1 PR, [s]
  u1 cs;            // Checksum
};
```

These messages contain corresponding relative pseudoranges for all the satellites specified in the latest [SI] message. The [1R], [2R], [3R], and [5R] messages contain relative P/L1, P/L2, CA/L2, and L5 pseudoranges, respectively.

Use the following formula to restore true pseudorange in seconds:

$$pr = rpr + prCA1$$

where prCA1 is corresponding CA/L1 pseudorange.

[1r], [2r], [3r], [5r]: Short Relative Pseudoranges

```
struct SRPR {2*nSats+1} {
! i2 srpr[nSats];  // (PR[s] - CA/L1 PR[s] - 2^-7) * 10^11
  u1 cs;           // Checksum
};
```

These messages are the most compact pseudorange format. They contain corresponding short relative pseudoranges for all the satellites specified in the latest [SI] message. The [1r], [2r], [3r], and [5r] messages contain short relative P/L1, P/L2, CA/L2, and L5 pseudoranges, respectively.

Use the following formula to restore true pseudorange in seconds:

$$pr = srpr * 10^{-11} + 2*10^{-7} + prCA1$$

where prCA1 is corresponding CA/L1 pseudorange.

[CC], [C1], [C2], [C3], [C5]: Smoothing Corrections

```
struct SC {6*nSats+1} {
! Smooth smooth[nSats];  // PR smoothing
  u1 cs;                 // Checksum
};
```

where “Smooth” format is defined as follows:

```
struct Smooth {6} {
! f4 value;     // Smoothing correction [s]
! u2 interval;  // Smoothing interval [s]
};
```

These messages contain corresponding pseudorange smoothing corrections and corresponding smoothing intervals for all the satellites specified in the latest [SI] message.

The [CC], [C1], [C2], [C3], and [C5] messages contain CA/L1, P/L1, P/L2, CA/L2, and L5 smoothing corrections, respectively.

Use the following formula to compute smoothed pseudoranges in seconds:

$$pr\_sm = pr + smooth.value$$

[cc], [c1], [c2], [c3], [c5]: Short Smoothing Corrections

```
struct SS {2*nSats+1} {
! i2 smooth[nSats];  // Smoothing correction [s*10^-11]
  u1 cs;             // Checksum
};
```

These messages contain corresponding short pseudorange smoothing corrections for all the satellites specified in the latest [SI] message. The [cc], [c1], [c2], [c3], and [c5] messages contain short CA/L1, P/L1, P/L2, CA/L2, and L5 smoothing corrections, respectively.

Use the following formula to compute smoothed pseudoranges in seconds:

$$pr\_sm = pr + smooth * 10^{-11}$$

[PC], [P1], [P2], [P3], [P5]: Carrier Phases

```
struct CP {8*nSats+1} {
! f8 cp[nSats];  // CP, [cycles]
  u1 cs;         // Checksum
};
```

These messages contain corresponding carrier phases for all the satellites specified in the latest [SI] message. The [PC], [P1], [P2], [P3], and [P5] messages contain CA/L1, P/L1, P/L2, CA/L2, and L5 carrier phases, respectively.

[pc], [p1], [p2], [p3], [p5]: Short Carrier Phases

```
struct SCP {4*nSats+1} {
! u4 scp[nSats];  // CP, [cycles/1024]
  u1 cs;          // Checksum
};
```

Note: The “scp” field will have discontinuities due to rollovers. Refer to “Compensating for Phase Rollovers” on page 369 for details.

These messages contain corresponding short carrier phases for all the satellites specified in the latest [SI] message. The [pc], [p1], [p2], [p3], and [p5] messages contain short CA/L1, P/L1, P/L2, CA/L2, and L5 carrier phases, respectively.

Use the following formula to compute full carrier phases in cycles:

$$cp = scp / 1024.0$$

[CP], [1P], [2P], [3P], [5P]: Short Relative Carrier Phases

```
struct RCP_RC {4*nSats+1} {
! f4 rcp[nSats];  // cp / F_Ln - prRC [s]
  u1 cs;          // Checksum
};
```

Thise messages contain differences between the full corresponding carrier phases and the matching [RC] pseudoranges for all the satellites specified in the latest [SI] message. The [CP], [1P], [2P], [3P], and [5P] messages contain CA/L1, P/L1, P/L2, CA/L2, and L5 carrier phases, respectively.

Use the following formula to compute true carrier phases in cycles:

$$cp = (rcp + prRC) * F_{Ln}$$

where:

prRC – is the value taken from corresponding [RC] message,

F_{Ln} – is nominal Ln carrier frequency for corresponding satellite, e.g., nominal L2 frequency for [2P] and [3P] messages, and nominal L1 frequency for [CP] and [1P] messages.

[cp], [1p], [2p], [3p], [5p]: Short Relative Carrier Phases

```
struct RCP_rc {4*nSats+1} {
! i4 rcp[nSats];  // cp / F_Ln - pr_rc, [s*2^-40]
  u1 cs;          // Checksum
};
```

These messages contain the differences between the full corresponding carrier phases and the matching [rc] pseudoranges for all the satellites specified in the latest [SI] message. The [cp], [1p], [2p], [3p], and [5p] messages contain CA/L1, P/L1, P/L2, CA/L2, and L5 carrier phases, respectively.

Use the following formula to retrieve true carrier phases in cycles:

$$cp = (rcp * 2^{-40} + pr\_rc) * F_{Ln}$$

where:

pr_rc – is the value taken from corresponding [rc] message, and converted to seconds as specified in the description of the [rc] message.

F_{Ln} – is nominal Ln carrier frequency for corresponding satellite, e.g., nominal L2 frequency for [2p] and [3p] messages, and nominal L1 frequency for [cp] and [1p] messages.

[DC], [D1], [D2], [D3], [D5]: Doppler

```
struct DP {4*nSats+1} {
! i4 dp[nSats];  // DP [Hz*10^-4]
  u1 cs;         // Checksum
};
```

These messages contain corresponding doppler estimates for all the satellites specified in the latest [SI] message. The [DC], [D1], [D2], [D3], and [D5] messages contain CA/L1, P/L1, P/L2, CA/L2, and L5 doppler, respectively.

Use the following formula to compute true doppler:

$$doppler = dp * 10^{-4}$$

[1d], [2d], [3d], [5d]: Short Relative Doppler

```
struct SRDP {2*nSats+1} {
! i2 srdp[nSats];  // (dp - dpCA1)[Hz*10^-4]
  u1 cs;           // Checksum
};
```

These messages contain corresponding short doppler relative to CA/L1 doppler for all the satellites specified in the latest [SI] message. The [1d], [2d], [3d], and [5d] messages contain P/L1, P/L2, CA/L2, and L5 short relative doppler, respectively.

Use the following formula to compute true doppler:

$$doppler = (srdp + dpDC) * F_{Ln} / F_{L1} * 10^{-4}$$

where:

`dpDC` – is the value `dp` taken from the [DC] message for given SV,

F_{L1} – is the nominal L1 frequency of the corresponding satellite,

F_{Ln} – is the nominal Ln frequency of the corresponding satellite, e.g., nominal L2 frequency for [2d] and [3d] messages.

[EC], [E1], [E2], [E3], [E5]: Carrier to Noise Ratio

```
struct CNR {nSats+1} {
! u1 cnr[nSats];  // C/N0 [dB*Hz]
  u1 cs;          // Checksum
};
```

Thise messages contain corresponding channel carrier to noise ratios for all the satellites specified in the latest [SI] message. The [EC], [E1], [E2], [E3], and [E5] messages contain CA/L1, P/L1, P/L2, CA/L2, and L5 carrier to noise ratio, respectively.

[CE], [1E], [2E], [3E], [5E]: Carrier to Noise Ratio x 4

```
struct CNR_4 {nSats+1} {
! u1 cnrX4[nSats];  // C/N0 [0.25*dB*Hz]
  u1 cs;            // Checksum
};
```

Thise messages contain corresponding channel carrier to noise ratios for all the satellites specified in the latest [SI] message. The [CE], [1E], [2E], [3E], and [5E] messages contain CA/L1, P/L1, P/L2, CA/L2, and L5 carrier to noise ratio multiplied by 4, respectively.

Use the following formula to compute true carrier to noise ratio in dB*Hz:

$$cnr = cnrX4 * 0.25$$

[FC], [F1], [F2], [F3], [F5]: Signal Lock Loop Flags

```
struct Flags {2*nSats+1} {
  u2 flags[nSats];  // Lock Loop Flags [bitfield]
  u1 cs;            // Checksum
};
```

Thise messages contain an array of corresponding signal lock loop flags for all the satellites specified in the latest [SI] message. The [FC], [F1], [F2], [F3], and [F5] messages contain CA/L1, P/L1, P/L2, CA/L2, and L5 signal lock loop flags, respectively.

The following flags are defined:

bit#	Hex Mask	Description
0	0x0001	PLL is in phase lock
1	0x0002	Satellite signal strength is sufficient
2	0x0004	Satellite uses guiding data from the common loop
3	0x0008	Satellite data are used in the common loop, which in turn controls this satellite's loops
4	0x0010	DLL is in steady state phase lock
5	0x0020	Loss-of-lock occurred in PLL between the previous and the current epochs
6	0x0040	Integral indicator: all data are good
7	0x0080	Not Used
8	0x0100	Preamble detected
9…15	0xFE00	Reserved

Note: bit#5 is not suitable for loss-of-lock detection in applications, – use [TC] message for this purpose instead.

The simplest approach to data validation is keeping track of only bit #6. As long as this bit remains set for a particular satellite, all of this satellite's measurements for corresponding signal type are considered good. Note that the receiver normally utilizes very narrow DLL bandwidths, thus, quite a long settling-down time (tens of seconds). In fact, it is not worth waiting until the pseudorange noise error reaches its steady-state level.

Note that bit#6 is set as soon as the measured pseudoranges become "accurate enough"[8] (i.e., irrespective of whether the formal settling-down period is over or not). On the other hand, for code differential applications, pseudorange accuracy is of critical importance. If bit#4 is set, this indicates that the corresponding pseudoranges are generated after the loop having reached the "steady state" and therefore are considered the least noisy.

In fact, it is not infrequent that raw data are used even if bit #6 is not set. In such cases, however, all responsibility for providing valid results rests with the user.

Bits ##0…3 are used for internal purposes.

[TC] CA/L1 Continuous Tracking Time

```
struct TrackingTimeCA {2*nSats+1} {
  u2 tt[nSats];  // tracking time [s]
  u1 cs;         // Checksum
};
```

This message contains time elapsed since the last loss-of-lock on the CA/L1 signal for every satellite specified in the latest [SI] message.

[TC] time is measured in seconds. Each satellite is allocated its own TC-time counter. Count-up begins with zero and stops when the counter reaches the maximum value the "u2" data type allows. Please note that the TC-time counters are not subject to rollovers.

Given a satellite, TC-time count starts as soon as the C/A signal is locked on. Should a loss of lock occur when tracking the C/A signal, the TC-time counter is reset to zero.

[SS] Satellite Navigation Status

```
struct NavStatus {nSats+2} {
  u1 ns[nSats];  // Navigation Status
  u1 solType;    // Solution type
  u1 cs;         // Checksum
};
```

This message contains navigation status for all the satellites specified in the latest [SI] message. In addition, this message indicates which receiver positioning mode the status belongs to. For detailed information on the navigation status, see Table 4-4, "Satellite Navigation Status," on page 287.

8. Say, at a level of unmodeled pseudorange errors.

[ID] Ionospheric Delays

```
struct IonoDelay {4*nSats+1} {
! f4 delay[nSats];  // Ionospheric delay [s]
  u1 cs;            // Checksum
};
```

This message contains estimated ionospheric delays as computed by using the L1 minus L2 frequency combination.

4.4.7 Almanacs and Ephemeris

[GA] GPS Almanac

```
struct GPSAlm {47} {
  u1 sv;        // SV PRN number within the range [1…37]
  i2 wna;       // Almanac reference week []
  i4 toa;       // Almanac reference time of week [s]
  u1 healthA;   // Health summary (from almanac), [bitfield]
                //   0…4 - code for health of SV signal components
                //   5…7 - navigation data health indicators
  u1 healthS;   // Satellite health (page 25 of subframe 5) []
  u1 config;    // Satellite configuration (page 25 of subframe 4)
                //   [bitfield]:
                //   0…2 - satellite configuration
                //     3 - anti-spoofing flag
                //   4…7 - reserved
  //======= Clock data =======
  f4 af1;       // Polynomial coefficient [s/s]
  f4 af0;       // Polynomial coefficient [s]
  //===== Ephemeris data =====
  //--- Keplerian orbital parameters ---
  f4 rootA;     // Square root of the semi-major axis [m^0.5]
  f4 ecc;       // Eccentricity []
  f4 m0;        // Mean Anomaly at reference time [semi-circles]
  f4 omega0;    // Longitude of ascending node of orbit plane
                //   at the start of week 'wna' [semi-circles]
  f4 argPer;    // Argument of perigee [semi-circles]
  //--- Corrections to orbital parameters ---
  f4 deli;      // Correction to inclination angle [semi-circles]
  f4 omegaDot;  // Rate of right ascension [semi-circle/s]
  u1 cs;        // Checksum
};
```

[EA] GALILEO Almanac

```
struct GALAlm {49} {
  // GPS-alike data
  GPSAlm gps; // Without 'cs' field, gps.sv within the range [1…30]
  // GALILEO-Specific data
  i2 iod;      // Issue of almanac data []
  u1 cs;       // Checksum
};
```

[NA] GLONASS Almanac

```
struct GLOAlmanac {46} {
  u1 sv;      // Satellite orbit slot number within [1…24] []
  i1 frqNum;  // Satellite frequency channel number [-7…24] []
  i2 dna;     // Day number within 4-year period starting
              //   with the leap year []
  f4 tlam;    // Time of the first ascending node passage
              //   on day 'dna' [s]
  u1 health;  // Satellite health as specified by 'Cn' [bitfield]
              //     0 - 1 indicates healthy SV, 0 - unhealthy
              //   1…7 - reserved
  //======= Clock data =======
  f4 tauN;    // Coarse time correction to SV clock
              //   with respect to GLONASS system time [s]
  f8 tauSys;  // Correction to GLONASS system time with respect
              //   to UTC(SU) [s]
  //===== Ephemeris data =====
  f4 ecc;     // Eccentricity at reference time 'tlam' []
  f4 lambda;  // Longitude of ascending node
              //   at reference time 'tlam' [semi-circles]
  f4 argPer;  // Argument of perigee
              //   at reference time 'tlam' [semi-circles]
  f4 delT;    // Correction to mean Draconic period
              //   at reference time 'tlam' [s/period]
  f4 delTdt;  // Rate of change of Draconic period [s/period^2]
  f4 deli;    // Correction to inclination
              //   at reference time 'tlam'[semi-circles]
  u1 cs;      // Checksum
};
```

[WA] WAAS Almanac

```
struct WAASAlmanac {23} {
  u1 waasPrn;          // WAAS SV PRN number within [120…138]
  u1 gpsPrn;           // GPS SV PRN associated with WAAS SV
  u1 id;               // Data ID
  u1 healthS;          // Satellite health [bitfield]:
                       //     0 - 0-Ranging on, 1-off
                       //     1 - 0-Corrections on, 1-off
                       //     2 - 0-Broadcast Integrity on, 1-off
                       //     3 - reserved
                       //   4…7 - are set to zero
  u4 tod;              // Time of the day [s]
  i4 xg, yg, zg;       // ECEF coordinates [m]
  f4 vxg, vyg, vzg;    // ECEF velocity [m/s]
  u4 tow;              // time of GPS week almanac was received at
  u2 wn;               // GPS week this almanac was received at
  u1 cs;               // Checksum
};
```

[GE] GPS Ephemeris

```
struct GPSEphemeris {123} {
  u1 sv;     // SV PRN number within the range [1…37]
  u4 tow;    // Time of week [s]
  u1 flags;  // Flags (see GPS ICD for details)[bitfield]:
             //   0 - curve fit interval
             //   1 - data flag for L2 P-code
             // 2,3 - code on L2 channel
             //   4 - anti-spoof (A-S) flag (from HOW)
             //   5 - 'Alert' flag (from HOW)
              //   6 - ephemeris was retrieved from non-volatile memory
             //   7 - reserved
  //===== Clock data (Subframe 1) =====
  i2 iodc;      // Issue of data, clock []
  i4 toc;       // Clock data reference time [s]
  i1 ura;       // User range accuracy []
  u1 healthS;   // Satellite health []
  i2 wn;        // Week number []
  f4 tgd;       // Estimated group delay differential [s]
  f4 af2;       // Polynomial coefficient [s/(s^2)]
  f4 af1;       // Polynomial coefficient [s/s]
  f4 af0;       // Polynomial coefficient [s]
  //===== Ephemeris data (Subframes 2 and 3) =====
  i4 toe;       // Ephemeris reference time [s]
  i2 iode;      // Issue of data, ephemeris []
  //--- Keplerian orbital parameters ---
  f8 rootA;     // Square root of the semi-major axis [m^0.5]
  f8 ecc;       // Eccentricity []
  f8 m0;        // Mean Anomaly at reference time (wn,toe)
                //   [semi-circles]
  f8 omega0;    // Longitude of ascending node of orbit plane at the
                //   start of week 'wn' [semi-circles]
  f8 inc0;      // Inclination angle at reference time [semi-circles]
  f8 argPer;    // Argument of perigee [semi-circles]
  //--- Corrections to orbital parameters ---
  f4 deln;      // Mean motion difference from computed value
                //   [semi-circle/s]
  f4 omegaDot;  // Rate of right ascension [semi-circle/s]
  f4 incDot;    // Rate of inclination angle [semi-circle/s]
  f4 crc;       // Amplitude of the cosine harmonic correction term
                //   to the orbit radius [m]
  f4 crs;       // Amplitude of the cosine harmonic correction term
                //   to the orbit radius [m]
  f4 cuc;       // Amplitude of the cosine harmonic correction term
                //   to the argument of latitude [rad]
  f4 cus;       // Amplitude of the cosine harmonic correction term
                //   to the argument of latitude [rad]
  f4 cic;       // Amplitude of the cosine harmonic correction term
                //   to the angle of inclination [rad]
  f4 cis;       // Amplitude of the sine harmonic correction term
                //   to the angle of inclination [rad]
  u1 cs;        // Checksum
};
```

[NE] GLONASS Ephemeris

```
struct GLOEphemeris {80} {
  u1 sv;       // Satellite orbit slot number [1…24] []
  i1 frqNum;   // Satellite frequency channel number [-7…24] []
  i2 dne;      // Day number within 4-year period []
  i4 tk;       // Frame start time within current day [s]
  i4 tb;       // Ephemeris reference time (for day 'dne') [s]
  u1 health;   // Satellite health [bitfield]
               //     0 - MSB taken from Bn word which indicates
               //         satellite health:
               //           1 - satellite is unhealthy
               //           0 - satellite is healthy
               //     1 - If set, this flag indicates that params
               //         'tau' and 'gamma' may be wrong
               //         (Note that receiver performs several
               //         'internal' data consistency checks allowing
               //         detection of problem broadcast parameters)
               //     2 - If set, this flag indicates that initial
               //         conditions 'r[3]' and 'v[3]' may be wrong
               //     3 - SV health (Cn word) status from almanac:
               //         0 - satellite is unhealthy
               //         1 - satellite is healthy
               //     4 - If set, this flag indicates that SV health
               //         status from almanac is available
               //   5…7 - reserved
  //===== Ephemeris data ======
  u1 age;      // Age of operational information (En) [days]
  u1 flags;    // Flags (for details on the flags, see GLONASS ICD)
               //   [bitfield]:
               //   0…1 - p1 word
               //     2 - p2 word
               //     3 - p3 word
               //   4…5 - 2 LSB taken from Bn word
               //     6 - ephemeris was retrieved from non-volatile
               //         memory
               //     7 - reserved
  f8 r[3];     // Satellite PE-90 coordinates [km]
  f4 v[3];     // Satellite PE-90 velocities [km/s]
  f4 w[3];     // Satellite PE-90 accelerations due to Luni-Solar
               //   gravitational perturbations [km/s^2]
//===== Clock data ======
  f8 tauSys;   // Time correction to GLONASS time scale (vs. UTC(SU))
               //   tauSys = TUTC(SU) - TGLN [s]
  f4 tau;      // Correction to satellite clock (vs. GLONASS time)
               //   tau = TGLN - TSV [s]
  f4 gamma;    // Rate of satellite clock offset [s/s]
  u1 cs;       // Checksum
};
```

[WE] WAAS Ephemeris

```
struct WAASEhemeris {39} {
  u1 waasPrn;           // WAAS SV PRN number within [120…138]
  u1 gpsPrn;            // GPS SV PRN associated with WAAS SV
  u1 iod;               // Issue of data
  u1 acc;               // WAAS SV accuracy9
  u4 tod;               // Reference time (seconds of the day)[s]
  f8 xg, yg, zg;        // ECEF coordinates [m]
  f4 vxg, vyg, vzg;     // ECEF velocity [m/s]
  f4 vvxg, vvyg, vvzg;  // ECEF acceleration [m/s^2]
  f4 agf0;              // WAAS SV clock offset factor 'ao' [s]
  f4 agf1;              // WAAS SV clock offset factor 'a1' [s/s]
  u4 tow;               // Time of GPS week this ephemeris was
                        //   received at
  u2 wn;                // GPS week this ephemeris was received at
  u1 cs;                // Checksum
};
```

[EN] GALILEO Ephemeris

```
struct GALEphemeris {144} {
  // GPS-alike data
  GPSEphemeris gps; // Without 'cs' field, gps.sv within the range [1…30]
  // GALILEO-specific data
  f4 bgdE1E5a; // broacast group delay E1 - E5A [s]
  f4 bgdE1E5b; // broacast group delay E1 - E5B [s]
  f4 ai0;      // Effective ionisation level 1-st order parameter []
  f4 ai1;      // Effective ionisation level 2-nd order parameter []
  f4 ai2;      // Effective ionisation level 3-rd order parameter []
  u1 sfi;      // Ionospheric disturbance flags [bitfield]
  u1 cs;       // Checksum
};
```

4.4.8 Raw Navigation Data

[GD] GPS Raw Navigation Data

```
struct GpsNavData {N*recSize+2} {
  u1     recSize;  // Size of satellite data record (currently 42)
  SvData dat[N];   // Satellite data. "N" can be derived from the
                   // following expression:
                   // N=([Message Length] - 2) / recSize)
  u1     cs;       // Checksum
};
```

9. For details, see ICD-GPS-200C, Revision IRN-200C-004 April 12, 2000.

```
struct SvData {recSize} {
  i1 fcn;           // Pseudo-Range Number (PRN)
  u1 cnt;           // Counter which is updated upon receiving a
                    // new sub-frame for given satellite.
  i4 data[10];      // GPS sub-frame contents. Every 4-bytes word
                    // contains 30 LSB of the GPS navigation data.
};
```

[LD] GLONASS Raw Navigation Data

```
struct GloNavData {N*recSize+2} {
  u1     recSize;   // Size of satellite data record (currently 18)
  SvData dat[N];    // Satellite data. "N" can be derived from the
                    // following expression:
                    // N=([Message Length] - 2) / recSize)
  u1     cs;        // Checksum
};

struct SvData {recSize} {
  i1 fcn1;          // Frequency Channel Number plus 1(FCN+1)
  u1 cnt;           // Counter which is updated upon receiving a
                    // string of a GLONASS sub-frame for given FCN.
  i4 data[4];       // GLONASS string contents. Every 4-bytes word contains
                    // 25 LSB of the string of GLONASS sub-frame.
};
```

[WD] WAAS Raw Navigation Data

```
struct WAASRawMessage {40} {
  u1 prn;    // SV PRN number within the range [120…138]
  u4 time;   // Time of receiving of message [s]
  u2 reserv; // Reserved
  u1 data[32]; // Navigation data block
  u1 cs; // Checksum
};
```

In the `data` field of the message the most significant bit of the first byte corresponds to the first broadcasted 4-ms data symbol. The field contains data strarting from WAAS preamble up to and including WAAS checksum.

[ED] GALILEO Raw Navigation Data

```
struct GALRawMessage {len+8} {
  u1 prn;       // SV PRN number within the range [1…30]
  u4 time;      // Time of receiving of message [s]
  u1 type;      // Type of data:
                //   0 - GALILEO-E1B
                //   3 - GIOVE-E1B
  u1 len;       // Length of the navigation data block 'data'
  u1 data[len]; // Navigation data block
  u1 cs; // Checksum
};
```

In the `data` field of the message the most significant bit of the first byte corresponds to the first broadcasted symbol.

For GIOVE-E1B, the `data` field of the message contains navigation data starting from the “Res-1” field up to and including the ”Tail” field.

4.4.9 ADU Messages

[MR] Rotation Matrix

```
struct RotationMatrix {37} {
  u4 time;     // receiver time [ms]
  f4 q00, q01, q02, q12;  // components of the rotation matrix Q []
  f4 rms[3];   // estimated accuracy for three baseline vectors [m]
  u1 solType[3];  // solution type10 for three baseline vectors
  u1 flag;     // 0 - components of matrix Q are invalid, 1 - valid
  u1 cs;       // Checksum
};
```

[mr] Rotation Matrix and Vectors

```
struct RotationMatrixAndVectors {73} {
  u4 time;    // receiver time [ms]
  f4 q00, q01, q02, q12;  // components of the rotation matrix Q []
  f4 rms[3];  // estimated accuracy for three baseline vectors [m]
  u1 solType[3];  // solution type10 for three baseline vectors
  u1 flag;    // 0 - components of matrix Q are invalid, 1 - valid
  f4 bl0[3];  // baseline vector M-S0 in the current epoch[m]
  f4 bl1[3];  // baseline vector M-S1 in the current epoch[m]
  f4 bl2[3];  // baseline vector M-S2 in the current epoch[m]
  u1 cs;      // Checksum
};
```

[AR] Rotation Angles

```
struct RotationAngles {33} {
  u4 time;        // receiver time [ms]
  f4 p, r, h;     // angles [deg]
  f4 sp, sr, sh;  // angles RMS values [deg]
  u1 solType[3];  // base lines solution types
  u1 flag;        // quality flag
  u1 cs;          // Checksum
};
```

10. See Table 4-3, “Solution Types,” on page 286

4.4.10 Event Marker and PPS Messages

The event marker and PPS have their own reference time settings governed by corresponding parameters[11]. As a consequence, some of the event marker and PPS messages described below utilize the “time scale” field of the following format:

Table 4-7. Event Marker and PPS time scale

Value	Description
0	GPS system time
1	UTC(USNO). Universal Coordinated Time supported by the U.S. Naval Observatory
2	GLONASS system time
3	UTC(SU). Universal Coordinated Time supported by the State Time and Frequency Service, Russia
4…255	Reserved

[XA], [XB] External Event

```
struct ExtEvent {10} {
  i4 ms;          // ms part of event time tag
  i4 ns;          // ns part of event time tag
  u1 timeScale;   // time scale
  u1 cs;          // Checksum
};
```

The event time tag is the time in corresponding time scale modulo one day.

To make your receiver generate these messages, you additionally need to turn on external event processing on corresponding external event input (using `/par/dev/event/[a|b]/in` parameters).

[ZA], [ZB] PPS Offset

```
struct PPSOffset {5} {
  f4 offs;  // PPS offset in nanoseconds
  u1 cs;    // Checksum
};
```

Due to a hardware limitation, PPS signals are discrete with a resolution of 25 ns. JAVAD GNSS receiver allows you to compensate for this discreteness error by means of utilizing this message. It contains the offset between the scheduled PPS time and the actual pulse edge's arrival time. When the pulse edge is earlier than the scheduled time, the offset is positive. When the pulse edge is delayed relative to the scheduled time, the offset is negative.

11. `/par/dev/event/[a|b]/time`, and `/par/dev/pps/[a|b]/time`

[YA], [YB] Time Offset at PPS Generation Time

```
struct RcvTimeOffsAtPPS {10} {
  f8 offs;        // [Tpps-Tr] offset[s]
  u1 timeScale;   // time scale
  u1 cs;          // Checksum
};
```

This message contains PPS reference time to receiver time offset at the moment of PPS generation.

4.4.11 Heading and Pitch Messages

[ha] Heading and Pitch

```
struct {
  f4 heading;   // Heading of the baseline between the base and the
                // rover receiver [radians]
  f4 pitch;     // Pitch of the baseline between the base and the
                // rover receiver [radians]
  u1 cs;        // Checksum
};
```

This message contains heading and pitch calculated by the RTK engine.

[RO] Lever Arm Cartesian Position

This message contains the position of the master antenna corrected by the rotated lever arm vector. It has exactly the same format as the [PO] message described on page 292.

[RG] Lever Arm Geodetic Position

This message contains the position of the master antenna corrected by the rotated lever arm vector. It has exactly the same format as the [PG] message described on page 293.

4.4.12 Integrated Messages

For the users that prefer to have different yet logically related data in a single message, the JAVAD GNSS receiver supports a set of integrated messages. For example, the message [rM] may contain all of the code and carrier phase measurements available in the receiver for the given epoch, though this is achieved in exchange for much more complex internal message structure. Integrated messages are also somewhat optimized for real-time applications, so we also sometimes call them “real-time messages”. The exact contents of these messages are defined by corresponding receiver parameters[12] that are not local to particular output stream. It means that using the integrated messages, one

can't have different variants of these messages to be enabled to be output to different output streams simultaneously.

In the integrated messages, the field "sample" serves two main purposes. First, it allows the user to preserve data integrity since messages referenced to a specific epoch will all have the same sample number. Second, this field allows the user to keep track of the number of lost messages issued through a given port since the sample number is incremented when the next epoch starts.

Integrated messages can be received by the user in an arbitrary order of precedence. Before decoding a message its CRC-16 checksum must be checked. Remember that the checksum is computed for all the bytes starting from the header of the message up to the checksum positions exclusive (for more information about CRC-16, please refer to "Computing CRC16" on page 365).

The following tables, which are given for user reference, will explain the relationships between the integrated messages, [rE], [rM], [rV], and the basic JPS messages.

[rM] can be used in place of the following messages:

Pseudorange measurements	[RC], [rc], [R1], [r1], [1R], [1r], [R2], [r2], [2R], [2r]
Carrier Phase measurements	[PC], [pc], [CP], [cp], [P1], [p1], [1P], [1p], [P2], [p2], [2P], [2p]
Doppler	[DC], [D1], [D2]
Signal Lock Loop Flags	[FC], [F1], [F2]
Carrier to Noise ratio	[EC], [E1], [E2]
Satellite navigation status	[SS]
Satellite Indices	[SI]
GLONASS Satellite System Numbers	[NN]
Receiver Date and Receiver Time	[RD], [~~]
Time since Last Loss-of-Lock[1]	[TC]
Receiver Reference Time to Receiver Time Offset[2]	[TO]

1. Note that there is a limitation on the maximum tracking time reported in [rM] (102.3 seconds).
2. Remember that there is a limitation on the clock offset resolution (125 ns).

[rV] can be used in place of the following messages:

Position/Velocity messages	[PO], [VE], [PV]
Solution time tag	[ST]

12. See "Parameters of Integrated GREIS Messages" on page 186.

[rE] can be used in place of the following messages:

Receiver Date and Receiver Time	[RD], [~~]

You can govern the structure of your [rM] message by means of parameters from section "Parameters of Integrated GREIS Messages" on page 186. Also note that the format of [rM] allows addition of new fields to the structure if necessary.

In the event of new fields showing up in the message, its version number is incremented of course. Note that lengths of the structures "Header" and "SlotRec" are specified in the message explicitly, which makes it possible to maintain backward compatibility with any older software using the message.

The message [rE], which was conceived as a time tag for any other message type, is reserved for future use. The field "sample", which will exist in any integrated message, is intended to maintain data integrity, i.e., all messages associated with a given epoch must have identical "sample" numbers.

[rE] Reference Epoch

```
struct RefEpoch {10} {
  u2 sample;   // Sample number [dimensionless]
  u2 scale;    // Time scale ID, leap second status and
               // week/day part of epoch representation [bitfield]
               //   15…13: time scale ID:
               //          0 - GPS, 1 - GLONASS, 2 - UTC;
               //   12…11: leap second status:
               //          0 - no leap second epoch;
               //          1 - positive leap second;
               //          2 - negative leap second;
               //          3 - leap second status is unknown;
               //       this flag shows whether a leap second
               //       occurred at the current epoch;
               //    10…0: week/day representation:
               //   (a) if time scale ID is GPS:
               //       week number [0…1023],
               //       1024 indicates unknown week number;
               //   (b) if time scale ID is GLONASS:
               //       day number within 4-year period [1…1461],
               //       0 indicates unknown day number
               //   (c) if time scale ID is UTC:
               //     day number within the year [1…366],
               //     0 indicates unknown day number;
  u4 reftime;  // Milliseconds part of epoch representation [ms]:
               //   (a) if time scale ID is GPS:
               //       milliseconds of GPS week;
               //   (b) if time scale ID is GLONASS:
               //       milliseconds of GLONASS day;
               //   (c) if time scale ID is UTC,
               //       milliseconds of UTC day;
  u2 crc16;    // 16-bit CRC
};
```

[rM] Raw Measurements

```
struct RawMeas {N*((14|10|6)*M+6)+14} {
  u2 sample;   // Sample number []
  u2 scale;    // See [rE] for description
  u4 reftime;  // See [rE] for description
  i2 clock;    // Clock offset:
               // 15…2: Clock offset
               //        -2^13 /+(2^13-1) [125 nanoseconds]:
               //  1…0: Clock offset ID:
               //        0 – clock offset is unavailable
               //        1 – [GPS - Receiver time]
               //        2 – [GLONASS - Receiver time]
               //        3 – reserved
  u2 flags;    // Flags [bitfield]:
               //   15…13: message version [0…7]
               //    12…8: total number of “svd” records (N)
               //     7…5: this value plus 6 makes the length
               //           of the structure “Header” in bytes
               //     4…0: this value plus 10 (for “version” 0 and 1),
               //           or 6 (for “version” [2…7]) makes the length
               //           of the structure “SlotRec” in bytes
  SvData svd[N]; // SVs data (see below)
  u2 crc16;      // 16-bit CRC
};

struct SvData {(14|10|6)*M+6} {
  Header header;    // Header (see below)
  SlotRec slot[M];  // Slot records (see below)
};

struct Header {6} {
  u4 refrange;      // Reference pseudo-range [0.02 meters]
  u1 usi;           // USI (see [SI] message)
  u1 num;           // Number of slot records (M) [bitfield]:
                    //   7…3: reserved
                    //   2…0: number of slot records minus one (M)
};

struct SlotRec {14|10|6} {
  // Note: The zeroth element of the array Slot[i],i=0,…,M-1,
  // unlike the other elements, does not contain corrections
  // to the reference pseudo-range from the Header structure.
  // To provide the user with additional information, the flag
  // ‘svst’ is used for ‘delrange’ in the zeroth slot.
  i2 svstOrDelrange;  // SV status [bitfield], or
                      // Delta pseudo-range [0.02 meters].
      // SV status [bitfield]:
      //   15…11: GLONASS slot number (for GPS SV the field
      //          is undefined), [0…24], 0 - unknown
      //    10…6: Channel number [0…31], 31 - unavailable
      //     5…0: SV navigation status
      //  Delta pseudo-range [0.02 meters]:
      //    [full pseudo-range for given slot]-[refrange]
  u4 word1;  // Packed data 1 [bitfield]:
             //   31…12: [carrier phase] - [refrange]
             //          [-219…(219-1)] [0.0005 meters]
             //    11…9: slot ID:
             //          0 - C/A L1; 1 - P1; 2 - P2; 3 - C/A L2;
```

```
              //          4 - L5; 5,6,7 - reserved
              //       8: reserved
              //       7: signal lock loop flags are available
              //       6: lock time is available
              //     5…0: Signal-to-noise ratio [dB*Hz]
  u2 flags;   // Signal lock loop flags (see [FC] message)
  u2 lock;    // Packed data 2 [bitfield]:
              //   15…12: fractional part of Signal-to-noise
              //          ratio [0.1 dB*Hz]
              //   11…10: reserved
              //     9…0: lock time [0.1 second]. Tracking time since
              //          last loss of lock. Varies between 0 and
              //          102.3 seconds. "Gets stuck" at 102.3s after
              //          the actual tracking time exceeds this value
              //          (until another loss of lock occurs).
  u4 word2;   // Packed data 3. Only present for "version" 0 [bitfield]:
              //   31…7: Doppler [-2^24…(2^24-1)], [0.001 Hz]
              //    6…0: reserved
};
```

When handling [rM] message, the following rules must be observed:

1. The user should retrieve from the message its version number and the lengths of the structures `Header` and `SlotRec`. These fields are necessary to maintain compatibility with older software in case the message structure is modified in the future. At present, there are three versions, 0, 1, and 2. Versions 1 and 2 are intended for RTK applications. In version 1, the field `word2` is removed from the structure `SlotRec` altogether, which results in a more compact data set as compared against version 0. In version 2, the fields `flags`, `lock`, and `word2` are removed from the structure `SlotRec`.
2. Next, the user should retrieve the "total number of `svd` records" field. Although it is possible to decode the message by using the message length from the message's header, taking into account the "total number of `svd` records" field simplifies the decoding.
3. Field `refrange` from the structure `SvData` serves as a reference for all the other code and carrier phase measurements available for the given satellite. In other words, all the measurements other than the reference pseudo-range, are represented as deltas referenced to a common reference value. Such an approach allows the reduction of message length. The first field in the structure `SlotRec` should be handled depending on whether the structure's "slot number" is zero or not.
4. Field `num` from the structure `Header` shows the total number of slot records (see the structure "`SlotRec`"). For example, if only C/A measurements are enabled in the message, the field `num` will be zero.

[rV] Receiver’s Position and Velocity

```
struct PosVelVector {42} {
  u2 sample;  // Sample number []
  u2 delta;   // [bitfield]:
              //   15…5: Difference between the raw measurement time
              //         (available from either [rM] or [rE] message)
              //         and the position time tag [-1024…1023], [5 ms]
              //    4…0: reserved
  u4 word1;   // 32 MSB of Position X-component;
  u4 word2;   // [bitfield]:
              //   31…24: 8 LSB of Position ECEF X-component [10-4 m]
              //          or Latitude [10-11 radians]
              //          or Grid (Local) X-component [10-4 m];
              //      23: 1 - indicates that Position is valid
              //   22…21: 0 - Position is given in ECEF system
              //          1 - Position is given in geodetic coordinates
              //              (latitude, longitude, height above
              //              ellipsoid)
              //          2 - Position is given in grid (or local)
              //              coordinates
              //          3 - reserved
              //   20…16: Number of GPS SVs used in computation;
              //      15: 1 - indicates that Velocity is valid
              //   14…13: reserved
              //    12…8: Number of GLONASS SVs used in computation;
              //     7…4: Position computation mode
              //          (see Table 4-3 on page 286);
              //     3…0: Velocity computation mode
              //          (see Table 4-3 on page 286);
  u4 word3;   //    31…0: 32 MSB of Position Y-component;
  u4 word4;   //   31…24: 8 LSB of Position ECEF Y-component [10-4 m]
              //          or Longitude [10-11 radians]
              //          or Grid (Local) Y-component [10-4 m];
              //   23…15: PDOP * 10 [];
              //    14…0: RMS velocity error [0.001 meters];
  u4 word5;   // 32 MSB of 40-bit ECEF position Z-component;
  u4 word6;   // [bitfield]:
              //   31…24: 8 LSB of 40-bit position ECEF Z-component
              //          or Height above ellipsoid [10-4 m];
              //   23…20: reserved;
              //    19…0: RMS Position error [0.001 m];
  u4 word7;   // [bitfield]:
              //   31…4: velocity X-component [10-4 m/s]
              //         or East component (if types 1 and 2 are
              //         selected in bits 22…21 of the ‘word2’ field);
              //    3…2: reserved;
              //    1…0: 2 MSB of GREIS datum number (see note below);
  u4 word8;   // [bitfield]:
              //   31…4: velocity Y-component [10-4 m/s]
              //         or North component (if types 1 and 2 are
              //         selected in bits 22…21 of the ‘word2’ field);
              //    3…0: bits 7…4 of datum number;
  u4 word9;   // [bitfield]:
              //   31…4: velocity Z-component [10-4 m/s]
              //         or North component (if types 1 and 2 are
```

```
                //         selected in bits 22…21 of the 'word2' field);
                //    3…0: 4 LSB of GREIS datum number;
  u2 crc16;     // 16-bit CRC;
};
```

Note: For GREIS datum numbers, please refer to *"Reference Ellipsoids and Local Datums"* available from *http://www.javad.com*. Currently GREIS datum numbers range between zero and 221.

[rT] Receiver Clock Offsets

```
struct ClockOffsets {var} {
  u2 sample;     // Sample number []
  u2 reserved;   // Reserved for future extensions
  u1 recSize;    // Size of data block, in bytes, that corresponds to
                 // the given satellite system (8 bytes currently);
  ClkOffs Offs[N];  // Clock offsets (see below).
        // 'N' can be derived from the following expression:
        // N = (len - 7) / recSize, where 'len' is message body
        // length taken from message header
  u2 crc16;         // 16-bit CRC
};

struct ClkOffs {
  u4 word1;   // [bitfield]:
              //     31: reserved;
              //     30: if set, improved timing mode is turned on;
              //   29…0: clock offset [10^-4 meters], bit combination
              //         0x20000000 means the clock offset is
              //         unavailable or exceeds ±536870911;
u4 word2;     // [bitfield]:
              //   31…29: reserved;
              //   28…26: navigation system (0 - GPS, 1 - GLN);
              //    25…0: derivative of clock offset [10^-4 m/s],
              //          bit combination 0x2000000 means that clock
              //          offset is unavailable or exceeds ±33554431;
};
```

4.4.13 Interactive Messages

Commands sent to the receiver may generate reply messages from the receiver. These human-readable text messages are output immediately as a response to corresponding commands. Interactive applications are the target for this class of messages.

[RE] Reply

```
struct RE {var} {
  a1 reply[];  // Reply
};
```

The contents of a reply message depends on what particular command has invoked this reply message (see Chapter 2 for more information about GREIS receiver commands and possible replies).

[ER] Error

```
struct ER {var} {
  a1 error[];  // Error description
};
```

If receiver gets a command that, for some reason, can't be executed, or produce an error during execution, then an error message is generated. The contents of the error message specifies what is wrong with the issued command.

4.4.14 Miscellaneous Messages

[IO] Ionospheric Parameters

```
struct IonoParams {39} {
  u4 tot;      // Time of week [s]
  u2 wn;       // Week number (taken from the first subframe)
  // The coefficients of a cubic equation representing
  // the amplitude of the vertical delay
  f4 alpha0;   // [s]
  f4 alpha1;   // [s/semicircles]
  f4 alpha2;   // [s/semicircles^2]
  f4 alpha3;   // [s/semicircles^3]
  // The coefficients of a cubic equation representing
  // the period of the model
  f4 beta0;    // [s]
  f4 beta1;    // [s/semicircles]
  f4 beta2;    // [s/semicircles^2]
  f4 beta3;    // [s/semicircles^3]
  u1 cs;       // Checksum
};
```

This message contains ionospheric correction parameters from GPS subframe 4, page 18. These parameters relate to an ionospheric model mainly used by single frequency GPS receivers.

For more information about this ionosphere model, please see ICD-GPS-200C, Revision IRN-200C-004 April 12, 2000.

[==](EV) Event

```
struct Event {var} {
  u4 time;     // Receiver time of event occurrence modulo day, [ms]
  u1 type;     // Event type (see below), []
  u1 data[];   // Event contents
  u1 cs;       // Checksum
};
```

This message is generated (if enabled) every time some event occurs in the receiver. Currently the following event types are defined:

0 – free-form event. Is generated by the “event” command (see “event” on page 46).

1 – firmware warning. The “data” field describes the warning in human-readable form.

[LT] Message Output Latency

```
struct Latency {2} {
  u1 lt;  // output latency [ms]
  u1 cs;  // Checksum
};
```

This message contains the difference between the actual output time of the first of the messages sent to the output stream at the given epoch, and this epoch's time-tag. Note that latency for an output stream may depend on the amount of messages requested to a different stream. For example, the more messages are output to port A, the bigger the latency of port B; this is because the receiver begins generation of messages for port B only after it has finished generating messages for port A.

[>>] Wrapper

```
struct Wrapper {var} {
  u1 id;           // Source identifier
  u1 data[size];   // Data from the source
  a1 cs[2];        // Checksum formatted as hexadecimal
};
```

This message is intended to wrap up arbitrary data. The size of the wrapped data (in bytes) is equal to the message length from the header less 3 (size=L-3).

This message is used for two different purposes:

1. To wrap data from an input stream that has been set to the “wrapped echo mode” (see /par/[port]/ewrap and /par/[port]/echo parameters). In this case it is generated whenever some data come to the stream. The “id” field then contains input stream identifier:

id	Source Stream
‘a’…’d’	serial ports A…D, /dev/ser/a…/dev/ser/d
‘A’…’E’	TCP ports A…E, /dev/tcp/a…/dev/tcp/d
‘U’	USB port A, /dev/usb/a
‘H’	Bluetooth port A, /dev/blt/a
‘N’	CAN port A, /dev/can/a
‘P’	TCP client port, /dev/tcpcl/a
‘V’	UDP port

2. To wrap arbitrary message(s) during periodic messages output, as specified by the `em` command for corresponding messages. In this case the `id` field is set to be numerically equal to (1 - `count`), where `count` is the field from the message scheduling parameters. See “Periodic Output” on page 18 and “em & out” on page 33 for details.

Note: This message is not subject to enabling/disabling using the `em` and `dm` commands. It is generated and output using its own rules.

[PM] Parameters

```
struct Params {var} {
  a1 params[];          // Parameters description
  a1 delim[2] = “,@”;   // Checksum delimiter
  a1 cs[2];             // Checksum formatted as hexadecimal
};
```

This message contains information on (most of) receiver parameters. When enabled, it will also be output every time one of the receiver parameters is changed.

Due to large number of parameters only part of a whole receiver parameter tree is output at every epoch, and multiple [PM] messages are typically output per epoch. In addition, a few starting [PM] messages containing values for specific parameters are output at the first receiver epoch after enabling the message.

The starting messages and messages generated at the time of updating of receiver parameters have the following format:

NAME=VALUE

where NAME denotes the parameter name, and VALUE denotes the parameter value.

The other messages have a slightly different format, specifically:

{[ITEM[,ITEM…]]}

where ITEM denotes either the value of a parameter, or a comma-separated list of ITEMs surrounded by braces.

[LH] Logging History

```
struct LoggingHistory {var} {
  u1 svsCount;         // Number of SVs
  u1 targetStream;     // Stream ID
  u2 issue;            // Issue of the history
  u2 bitsCount;        // Number of bits
  u4 lastBitTime;      // Time since the last history shift [ms]
  u1 uids[svsCount];   // SVs UIDs
  u1 pad[padCount];    // Padding
  u4 hist[elemsCount][svsCount];  // History bits
};
```

This message contains history of logging of satellites data into particular stream. For a description of how logging history works and parameters governing logging history, see , “Logging History” on page 186.

Fields description:

`svsCount` – Number of SVs in this history.

`targetStream` – The stream ID the history is gathered for (see description of the [>>] message for details).

`issue` – The issue of the history. It is incremented every time the history is changed.

`bitsCount` – Number of bits in this history. This history contains this number of bits for every SV specified in the `uids` field.

`lastBitTime` – Time in milliseconds since the last history shift.

`uids[svsCount]` – Array of SVs UIDs

`pad[padCount]` – Padding (with zeroes) to align the next field on 4 bytes boundary,

$$fillCount = (4 - (svsCount \% 4)) \% 4$$

`hist[elemsCount][svsCount]` – History bits. For every SV, the bits are packed into array of `u4` values. Most significant bit of the first element of the array represents most recent bit of the history. Least significant bit of the last element of the array represents the oldest bit of the history when there are enough history bits to fill the last `u4` element. The number of `u4` elements in the array is just enough to hold `bitsCount` bits:

$$elemsCount = (bitsCount + 31) / 32$$

Exactly `svsCount` bit arrays are put into the message in the order specified by the `uids` field.

[BI] Base Station Information

```
struct BaseInfo {28} {
  f8 x, y, z;  // ECEF coordinates [m]
  u2 id;       // Reference station ID
  u1 solType;  // Solution type
  u1 cs;       // Checksum
};
```

[SE] Security

```
struct Security {6} {
  u1 data[5];  // Opaque data
  u1 cs;       // Checksum
};
```

This message is for JAVAD GNSS internal use.

[SM] Security for [rM]

```
struct Security {8} {
  u1 data[6];  // Opaque data
  u2 crc16;    // 16-bit CRC
};
```

This message is for JAVAD GNSS internal use.

[TT] CA/L1 Overall Continuous Tracking Time

```
struct TrackingTime {5} {
  u4 tt;  // tracking time [s]
  u1 cs;  // Checksum
};
```

This message contains time elapsed since the last loss-of-lock of all CA/L1 signals. Time count starts as soon as the first CA/L1 signal is locked on. Should a loss of lock of the last CA/L1 signal occur, the time counter is reset to zero.

[OO] Oscillator Offset

```
struct RcvOscOffs {5} {
  f4 val;  // Oscillator offset [s/s]
  u1 cs;   // Checksum
};
```

The offset is the difference between oscillator frequency when oscillator control voltage has nominal value, and the nominal frequency of the oscillator, normalized to the nominal frequency of the oscillator. $(F_{Un} - F_n)/F_n$.

Note: The contents of this message are not suitable as a parameter for calculations based on receiver measurements. Use [DO] message instead.

[||](EE) Epoch End

```
struct EpochEnd {1} {
  u1 cs;   // Checksum
};
```

This message carries no information. It is intended as an “end of epoch” marker useful for real-time applications. This message is not recommended. We suggest to use the [::](ET) message instead, see “[::](ET) Epoch Time” on page 289.

4.4.15 Text Messages

GREIS Format for Text Messages

All the text messages have the following format:

```
struct Text {var} {
  a1 text[];
};
```

General format of the “text” field of GREIS predefined text messages as well as format notation used to specify particular text messages is described in this section.

The format of the “text” field is as follows:

```
,TITLE[,ITEM[,ITEM,…]],@CS
```

where square brackets designate optional parts,

`TITLE` – the title of particular message;

`ITEM` – either a field of particular type described below, or an item list surrounded by braces:

```
{[ITEM[,ITEM,…]]}
```

`CS` – checksum formatted as two hexadecimal uppercase digits.

A GREIS text message's format specifies its structure, field types, and the number of significant digits for each field.

Field format notation always starts with symbol “%” (hex 25).

The following data type characters (aka “data type specifiers”) are used to distinguish between different data types:

`D` – decimal integer;

`X` – hexadecimal integer;

`C` – character;

`S` – string type (note that strings may have arbitrary lengths);

`F` – floating point;

`E` – exponential format for floating point.

Given a numeric field, digits preceding the data type specifier designate the number of digits in the format of this field[13]. There may be two, one or no digits specified before the data type specifier. For floating point fields, the first digit defines the length of the integer part of the field representation (“integer part descriptor”) whereas the second

13. This applies to integer, float and double fields only.

digit defines the length of the fractional part (“fractional part descriptor”). If there are two digits used in the field format notation, these are separated by a dot “.” (hex 2E). A fields's fractional part descriptor can be variable, i.e., its length is programmable with an appropriate receiver command. In this case the fractional part descriptor shows the parameter range, which is put inside square brackets (e.g., %0.[1-4]F). Integer part descriptor may be omitted, which means that the integer part of the output value is allowed to be as long as necessary. Note that the delimiting dot before the corresponding fractional part descriptor will still exist.

If an integer field format has no integer part descriptor before its data type specifier (either “X” or “D”), the receiver will output all the significant digits. Leading zeroes will be added if the actual length of the integer part is shorter than specified by the descriptor.

If a format notation includes the plus symbol “+”, receiver will output signed values (as usual, characters “+” (hex 2B) and “-“(hex 2D) are used to denote positive and negative values, respectively).

If a field format is surrounded by round brackets, it means that this format applies to a batch of “homogeneous” fields (note that the number of fields in such a batch may vary).

A back slash followed by two hexadecimal digits designates that the text character with corresponding ASCII code will be put into the message in this specific place. In addition, the reader will notice that various non-reserved symbols (lower and upper case English letters, arithmetic operation signs, braces, etc.,) are used together with the above described field formats. Here are some examples illustrating the format notation as defined above:

Format	Output data representation
%5.2F	00027.89
%5.2F * %+4.2F	67283.67 * +5678.22
Value: %+5.2F	Value: +00000.35
%+2.0F	+07
%0.4F	%0.4F 0.1234
%+7.5F	-0000039.67432
5-2=%D	5-2=3
%D	2467
%+3D	-052
%+3D	+964
%2X	0A
{%2D,%+.4F}	{04,-45.8027}
%C%C%C	XYZ

%S	This is a string
\4A	J
\50	P
(%2.1F)	45.8,98.0,04.7,88.3
%.3E	1.234E-04
%.3E	-5.789E+02

[DL] Data Link Status

This message displays the status of the data links associated with the corresponding serial ports/modem.

#	Format	Description
1	DLINK	Message title
2	%1D	Total number of data links (0…5). The rest of the message is available only if this number is non-zero. Otherwise the total number of data links value is immediately followed by the checksum.
3	({%C,%C,%S,%3D,%4D,%4D,%.2F})	Group of fields associated with the given data link (note that the total number of such groups is determined by the previous field). These fields are - Data link identifier (refer to Table 4-8 below). - Decoder identifier ("R" – RTCM decoder, "T" – RTCM 3.0 decoder, "C" – CMR decoder, "J" – JPS decoder). - Reference station identifier. - Time [in seconds] elapsed since receiving last message (maximum value = 999). Estimated with an accuracy of ±1 second. - Number of received messages (between 0001 and 9999). If no message has been received, this data field contains zero. - Number of corrupt messages (between 0001 and 9999). If no corrupt messages have been detected, this data field is set to zero. - Data link quality in percent (0-100);
4	@%2X	Checksum

Table 4-8. Data Link Identifiers

Id	Corresponding Stream
'A'…'D'	serial ports A…D, `/dev/ser/a…/dev/ser/d`
'E'…'I'	TCP ports A…E, `/dev/tcp/a…/dev/tcp/d`
'P'	TCP client port, `/dev/tcpcl/a`
'U'	USB port A, `/dev/usb/a`
'L'	Bluetooth port A, `/dev/blt/a`
'g'	CAN port A, `/dev/can/a`

[GS] GPS SVs Status

This message describes the status of GPS satellites.

#	Format	Description
1	GPSVST	Message title
2	%2D	Total number of GPS SVs being track
3	({%2D,%2D,%3D, {%2D[,%2D…]},%2D})	This 5-field section comprises: 1. GPS PRN number 2. Elevation in degrees 3. Azimuth in degrees, 4. Signal-to-noise ratios in [dB*Hz]. This is a list of variable number of elements, where elements are always in the following order: 1. CA/L1 signal-to-noise ratio 2. P/L1 signal-to-noise ratio 3. P/L2 signal-to-noise ratio 4. L2C signal-to-noise ratio 5. L5 signal-to-noise ratio 5. Channel navigation status (see Table 4-4 on page 287) The total number of such 5-field sections will match the number of SVs being track.
4	@%2X	Checksum

[ES] GALILEO SVs Status

This message describes the status of GALILEO satellites.

#	Format	Description
1	ESSVST	Message title
2	%2D	Total number of GALILEO SVs being track
3	({%2D,%2D,%3D, {%2D[,%2D…]},%2D})	This 5-field section comprises: 1. GALILEO PRN number 2. Elevation in degrees 3. Azimuth in degrees, 4. Signal-to-noise ratios in [dB*Hz]. This is a list of variable number of elements, where elements are always in the following order: 1. CA/L1 signal-to-noise ratio 2. Missed (reserved) 3. Missed (reserved) 4. Missed (reserved) 5. E5A signal-to-noise ratio 5. Channel navigation status (see Table 4-4 on page 287) The total number of such 5-field sections will match the number of SVs being track.

#	Format	Description
4	@%2X	Checksum

[WS] SBAS (WAAS/EGNOS) SVs Status

This message describes the status of SBAS satellites.

#	Format	Description
1	WSSVST	Message title
2	%2D	Total number of SBAS SVs being track
3	({%2D,%2D,%3D, {%2D[,%2D…]},%2D})	This 5-field section comprises: 1. SBAS PRN number 2. Elevation in degrees 3. Azimuth in degrees, 4. Signal-to-noise ratios in [dB*Hz]. This is a list of variable number of elements, where elements are always in the following order: 1. CA/L1 signal-to-noise ratio 2. Missed (reserved) 3. Missed (reserved) 4. Missed (reserved) 5. L5 signal-to-noise ratio 5. Channel navigation status (see Table 4-4 on page 287) The total number of such 5-field sections will match the number of SVs being track.
4	@%2X	Checksum

[NS] GLONASS SVs Status

This message describes the status of GLONASS satellites.

#	Format	Description
1	GLSVST	Message title
2	%2D	Total number of GLONASS SVs being track

#	Format	Description
3	({%2D,%2D,%2D,%3D,{%2D[,%2D…]},%2D})	This 6-field section comprises: 1. GLONASS SV Orbit Slot Number[1] 2. GLONASS SV Frequency Channel Number 3. Elevation in degrees 4. Azimuth in degrees 5. Signal-to-noise ratios in [dB*Hz]. This is a list of variable number of elements, where elements are always in the following order: 1. CA/L1 signal-to-noise ratio 2. P/L1 signal-to-noise ratio 3. P/L2 signal-to-noise ratio 4. CA/L2 signal-to-noise ratio 6. Channel navigation status (see Table 4-4 on page 287) The total number of such 6-field sections will match the number of SVs being track.
4	@%2X	Checksum

1.If orbit slot number is reported as zero, the slot number hasn't yet been determined.

[RS] Reference Station Status

This message contains parameters related to the reference station status.

#	Format	Description
1	REFST	Message title
2	%C	UTC time indicator: “V” means that UTC time is valid “N” means that UTC time is not valid
	%6.2F	UTC time of position (the first two digits designate hours, the next two digits designate minutes and the rest digits designate seconds)

#	Format	Description
4	{%C,%C,%C,%C}	Checking reference station location: - the first field relates to the reference station APC coordinates used for referencing GPS data (see /par/ref/pos/gps/… parameters), - the second field relates to the reference station APC coordinates used for referencing GLONASS data (see /par/ref/pos/glo/… parameters), - the third field relates to the reference station ARP coordinates used for referencing GPS data (see /par/ref/arp/gps/… parameters), - the fourth field relates to the reference station ARP coordinates used for referencing GLONASS data (see /par/ref/arp/glo/… parameters).[1] “V” – means that the difference between the current receiver coordinates and the user-defined reference coordinates does not exceed the specified limit; “N” – means that the difference between the current receiver coordinates and the user-defined reference coordinates is greater than the specified limit; (see “Maximum Allowed Error in Reference Position” on page 147)
5	@%2X	Checksum

1. Parameters mentioned above are described in the section “Reference Station Coordinates” on page 143.

[MS] RTCM 2.x Status

This message describes the status of RTCM rover station.

#	Format	Description
1	RTCMST	Message title
2	%3D	Time [in seconds] elapsed since last message was received (maximum value = 999). Estimated with an accuracy of ±1 second.
3	%4D	Number of received messages (between 0001…9999). If no message has been received, this data field contains zero.
4	%4D	Number of corrupt messages (between 0001…9999). If there are no corrupt messages detected, this data field is set to zero.
5	%.2F	Data link quality in percent (0…100).
6	@%2X	Checksum

Note: The [MS] is considered obsolete. We recommend that you use [DL] instead.

[TX] RTCM/CMR Text Message

This message allows the user to view text information derived on the rover end from messages RTCM 2.x Type 16, 36, 23, 24, RTCM 3.0 and CMR Type 2.

#	Format	Description
1	TEXT	Message title
2	%1D	Total number of the messages (0…2). The rest of the message is available only if this number is non-zero. Otherwise the total number of the messages value is immediately followed by the checksum.
3	({%02D/%02D,%02D:%02D:%02D,"%S",%C,%C,%S})	Group of fields associated with the given text message (note that the total number of such groups is determined by the previous field). These fields are - Date of receiving of the message (MM/DD - month/day). -UTC time of receiving of the message (HH:MM:SS) -String value of up to 90 characters. -Decoder identifier (see below) -Data link identifier ("A"…"D" – serial ports, "M" – modem). -Reference station identifier.
4	@%2X	Checksum

Decoder identifier:

- A – shows that information has been decoded from RTCM messages 23 and 24;
- R – shows that information has been decoded from RTCM messages 16 and 36;
- T – shows that information has been decoded from RTCM 3.0 messages 1007 and 1008;
- N – shows that information has been decoded from RTCM 3.0 message 4091 (Topcon proprietary message, `/msg/rtcm3/4091t`);
- C – CMR decoder
- J – GREIS messages decoder.

[RM] Results of RAIM Processing

This message contains RAIM output data.

#	Format	Description
1	RAIM	Message title

#	Format	Description
2	%C	UTC time indicator: "V" means that UTC time is valid "N" means that UTC time is not valid
3	%6.2F	UTC time of position (the first two digits designate hours, the next two digits designate minutes and the rest of the digits designate seconds)
4	%1D	RAIM indication: 0 - no anomalous measurements have been detected (position is valid); 1 - RAIM is not able to detect anomalous measurements (for example, due to poor geometry or limited number of visible satellites) – position may be badly affected by anomalous measurements; 2 - RAIM has detected and excluded anomalous measurements (position is valid). Note: Generally speaking, if more than one bad measurement is detected, there is no guarantee that the RAIM has excluded all bad measurements with the specified probability. However, in most cases, RAIM runs properly even if more than one bad measurement has been detected. If the RAIM indicator is other than "2", this data field is followed by the checksum). 3 – RAIM is turned off (position may be badly affected by poor measurements);
5	%D	Total number of excluded bad measurements
6	(%C%2D)	IDs of the satellites with bad measurements. Note that the total number of such satellites is determined by the previous field. - Navigation system identifier: "G" – designates GPS satellites, "R" or "F" – both designate GLONASS satellites. "F" is used for "R" until the receiver has determined the satellite's slot number. - Satellite system number (or, for GLONASS, satellite frequency channel number): GPS SV PRN (follows after the "G" flag), GLONASS SV slot number (follows after the "R" flag) or GLONASS SV frequency channel number (follows after the "F" flag);
7	@%2X	Checksum

[NP] Navigation Position

This message includes the receiver's navigational and positioning parameters.

#	Format	Description
1	NAVPOS	Message title
2	%C	UTC time indicator: "V" means that UTC time is valid "N" means that UTC time is not valid
3	%6.2F	UTC time of position (the first two digits designate hours, the next two digits designate minutes and the rest of the digits designate seconds)

#	Format	Description
4	%1D	Position computation indicator. If it equals “0”, this message contains more meaningful information (see the following data fields). If this indicator is non-zero, this data field is followed by the checksum). Position is valid only if the position computation indicator is equal to “0”. For how to interpret other values, see Table 4-9 on page 336.
5	%C%C	Position (first symbol) and velocity (second symbol) computation mode. See Table 4-10 on page 336.
6	{%2D,%2D}	Number of GPS and GLONASS satellites used in position computation
7	%S	Reference geodetic datum identifier
8	%C%2Do%2D ’%2.6F"	Latitude: hemisphere (“N” – northern, “S” – southern), degrees, minutes and seconds
9	%C%3Do%2D ’%2.6F"	Longitude: hemisphere (“E” – eastern, “W” – western), degrees, minutes and seconds
10	%+5.4F	Altitude above ellipsoid [meters]
11	%C	Geoidal separation indicator: “V” means geoidal separation is valid; “N” means geoidal separation is not valid.
12	%+.4F	Geoidal separation: the difference between the earth ellipsoid and geoid (mean-sea-level) defined by the reference datum [meters]
13	%.2F	Horizontal dilution of precision (HDOP) []
14	%.2F	Vertical dilution of precision (VDOP) []
15	%.3F	Horizontal position RMS error [meters]
16	%.3F	Vertical position RMS error [meters]
17	%.4F	Horizontal velocity [kilometers/hour]
18	%+.4F	Vertical velocity [kilometers/hour]
19	%3.3F	True heading [degrees]
20	%C	Magnetic heading indicator: “V” means magnetic heading is valid; “N” means magnetic heading is not valid.
21	%3.3F	Magnetic heading [degrees]
22	%.3F	Horizontal velocity RMS error [meters/second]
23	%.3F	Vertical velocity RMS error [meters/second]

#	Format	Description
24	%.2F	Data link quality in percent (0…100)
25	%3D	Time [in seconds] elapsed since last RTCM, CMR or JPS message was received (maximum value = 999). Estimated with an accuracy of ±1 second.
26	@%2X	Checksum

Table 4-9. Position Computation Indicator

0	Position is valid
1	Too many iterations have been made (position is not valid)
2	Singular matrix (position is not valid)
3	Not enough data for position computation (position is not valid)
4	Either or both altitude and speed exceed specified threshold values (position is not valid)
5	PDOP exceeds specified threshold value (position is not valid) See the parameter `/par/pos/pdop` in "Positioning Parameters" on page 83.
6	Wrong position. Calculated position is outside of sensible margins.
7	Position was computed, but the output of given type of solution is disabled by the user (e.g., stand-alone position is not output due to `/par/pos/mode/sp` parameter is set to `off`)

Table 4-10. Position/Velocity Computation Mode

A	Autonomous mode
D	Code differential mode
C	RTK positioning with codes
F	RTK positioning with float integers
R	RTK positioning with fixed integers
P	Fixed position, i.e., entered by user (not computed)

[MP] Position in Map Projection

The message describes receiver position in the specified map projection or local coordinate system.

#	Format	Description
1	MAPRJ	Message title

#	Format	Description
2	%C	UTC time indicator: "V" means that UTC time is valid "N" means that UTC time is not valid
3	%6.2F	UTC time of position (the first two digits designate hours, the next two digits designate minutes and the rest digits designate seconds)
4	%1D	Position computation indicator. If it equals "0", this message contains meaningful information (see the following data fields). If this indicator is non-zero, this data field is followed by the checksum. Position is valid only if the position computation indicator is equal to "0". For how to interpret other values, see Table 4-9 on page 336.
5	%C	Grid (or local) coordinates indicator: "V" means that Grid (or local) coordinates are valid "N" means that Grid (or local) coordinates are not valid
6	%S	Grid system ID (see Table below)
7	%2D	Zone of the grid system ("00", if not available)
8	%+.4F	Northern component of grid coordinates or "X" component of local coordinates [meters]
9	%+.4F	Eastern component of grid coordinates or "Y" component of local coordinates [meters]
10	%+5.4F	Altitude above ellipsoid or local horizon (for local coordinates) [meters]
11	@%2X	Checksum

Grid system IDs

UTM	Universal Transversal Mercator
TM	User-defined Transversal Mercator projection
STER	User-defined Stereographic projection
LOC	Local coordinates

[NR] Lever Arm Position

This message contains the position of the master antenna corrected by the rotated lever arm vector.

#	Format	Description
1	ARMPOS	Message title

#	Format	Description
2	%C	UTC time indicator: "V" means that UTC time is valid "N" means that UTC time is not valid
3	%6.2F	UTC time of position (the first two digits designate hours, the next two digits designate minutes and the rest of the digits designate seconds)
4	%1D	Position computation indicator. If it equals "0", this message contains more meaningful information (see the following data fields). If this indicator is non-zero, this data field is followed by the checksum). Position is valid only if the position computation indicator is equal to "0". For how to interpret other values, see Table 4-9 on page 336.
5	%C%C	Position (first symbol) and velocity (second symbol) computation mode (below). See Table 4-10 on page 336.
6	{%2D,%2D}	Number of GPS and GLONASS satellites used in position computation
7	%S	Reference geodetic datum identifier
8	%C%2Do%2D '%2.6F"	Latitude: hemisphere ("N" – northern, "S" – southern), degrees, minutes and seconds
9	%C%3Do%2D '%2.6F"	Longitude: hemisphere ("E" – eastern, "W" – western), degrees, minutes and seconds
10	%+5.4F	Altitude above ellipsoid [meters]
11	%.3F	Horizontal position RMS error [meters]
12	%.3F	Vertical position RMS error [meters]
13	@%2X	Checksum

[TR] Time Residuals

This message is intended for various time transfer applications. It contains information allowing the user to "match" an external clock to a specific GPS/GLONASS satellite's time scale.

#	Format	Description
1	TIMERES	Message title
2	%C	UTC time indicator: "V" means that UTC time is valid "N" means that UTC time is not valid

#	Format	Description
3	%6.2F	UTC time of position (the first two digits designate hours, the next two digits designate minutes and the rest digits designate seconds)
4	%2D	Total number of satellites used in position computation
5	({%C%2D,%.2F,%D})	Group of fields associated with the given satellite (note that the total number of such groups is determined by the previous field). These fields are - Navigation system identifier: “G” – designates GPS satellites, “R” or “F” – both designate GLONASS satellites. “F” is used for “R” until the receiver has determined the satellite's slot number. - Satellite system number (or, for GLONASS, satellite frequency channel number): GPS SV PRN (follows after the “G” flag), GLONASS SV slot number (follows after the “R” flag) or GLONASS SV frequency channel number (follows after the “F” flag); - “Time residual” in nanoseconds for the given satellite. This satellite-specific time correction is defined as V1 - V2, where - V1 is the system time[1] to the receiver time offset as estimated using this particular satellite (i.e., the difference between the satellite's own clock[2] and the receiver clock). - V2 is the system time to the receiver time offset as estimated using all of the satellites belonging to this navigation system. Note that it is V2 that governs PPS signals generated in the receiver. - Issue of data for satellite ephemeris: For GPS satellites this field includes IODE (Issue Of Data, Ephemeris); For GLONASS satellites this field includes Ephemeris reference time t_b (7 LSB) and the indicator (MSB), which is set to “1” if the ephemeris has been updated without a change in tb. Note: The interpretation of this field for GLONASS satellites is subject to change in the future.
6	@%2X	Checksum

1. “System time” means GPS and GLONASS system time for GPS and GLONASS satellites, respectively.
2. Assuming that the satellite clock has already been corrected to the corresponding system time, either GPS or GLONASS time, by applying the broadcast frequency-time corrections. The reader will notice that these “satellite-specific time residuals” first of all describe such effects as SA, multipath and atmospheric delay, which are all satellite-specific.

[TM] Clock Offsets and Time Derivatives

The message contains clock offsets (the difference between receiver time scale and GPS/GLONASS system time) and their derivatives.

#	Format	Description
1	TIMING	Message title
2	%C	UTC time indicator: “V” means that UTC time is valid “N” means that UTC time is not valid

#	Format	Description
3	%6.2F	UTC time of position (the first two digits designate hours, the next two digits designate minutes and the rest of the digits designate seconds)
4	%1D	Position computation indicator. If it equals "0", this message contains more meaningful information (see the following data fields). If this indicator is non-zero, this data field is followed by the checksum. Position is valid only if the position computation indicator is equal to "0". For how to interpret other values, see Table 4-9 on page 336.
5	%C	Position computation mode. See Table 4-10 on page 336.
6	{%2D,%2D}	Number of GPS and GLONASS satellites used in position computation
7	{{%C	GPS time fields indicator: "V" means that GPS time fields are valid "N" means that GPS time fields are not valid
8	%+.4F}	[GPS system time] - [Receiver time], [meters]
9	%+.4F}	Derivative of ([GPS system time] - [Receiver time]), [meters/second]
10	{%C	GLONASS time fields indicator: "V" means that GLONASS time fields are valid "N" means that GLONASS time fields are not valid
11	%+.4F	[GLONASS system time] - [Receiver time], [meters]
12	%+.4F}}	Derivative of ([GLONASS system time] - [Receiver time]), [meters/second]
13	%.2F	Time dilution of the precision (TDOP)
14	%.3F	Clock offsets RMS error [meters]
15	%.3F	RMS of the derivatives of clock offsets [meters/second]
16	%1D	The Improved Timing mode indicator. If it equals "0", this means the Improved Timing mode is turned off. If this field is "1" the mode is turned on.
17	@%2X	Checksum

[RP] Reference Station Parameters

This message contains the reference station parameters such as the station's coordinates, antenna offsets, station ID, etc. These parameters are used in RTK on the rover side.

These parameters are available via RTCM Messages Types 3, 22, 23, 24 or 31, RTCM 3.0, CMR Message Type 1 or JPS Message [BI].

#	Format	Description
1	REFPAR	Message title
2	%1D	Total number of groups containing the reference station fields. In the current version of the message, this field can be set to 0 or 1. “0” means that the reference station fields are not valid. “1” means that this message contains valid information.
3	({%S	Reference geodetic datum identifier
4	%C%2Dd%2D m%2.6Fs	Latitude: hemisphere (“N” – northern, “S” – southern), degrees, minutes and seconds
5	%C%3Dd%2D m%2.6Fs	Longitude: hemisphere (“E” – eastern, “W” – western), degrees, minutes and seconds
6	%.4F	Altitude above ellipsoid [meters]
7	%C	Antenna height indicator: “V” means antenna height is valid; “N” means antenna height is not valid.
8	%.4F	Antenna height [meters]
9	%C	Antenna East/North offsets indicator: “V” means East/North offsets are valid; “N” means East/North offsets are not valid.
10	%.4F	Antenna North offset [meters]
11	%.4F	Antenna East offset [meters]
12	%C	Decoder identifier, R, T, C, J, where A – shows that information has been decoded from RTCM messages 23 and 24; R – shows that information has been decoded from RTCM messages 3, 22 and 31; T – shows that information has been decoded from RTCM 3.0; C – CMR decoder J – GREIS messages decoder.
13	%C	Data link identifier (“A”…“D” – serial ports, “M” – modem, “U” – by user input).
14	%S	Reference station identifier.
15	%S})	Antenna ID. Two types of antenna IDs are supported: 1.IDs that are approved and standardized by NGS. 2.IDs that are used in the Trimble’s CMR format. CMR antenna IDs will always contain three digits.
16	@%2X	Checksum

Note: By default this message is output only after its contents have changed.

[RK] RTK Solution Parameters

This message contains some parameters of an RTK solution.

#	Format	Description
1	RTKPAR	Message title
2	%C	UTC time indicator: “V” means that UTC time is valid “N” means that UTC time is not valid
3	%6.2F	UTC time of the position fix (the first two digits designate hours, the next two digits designate minutes and the rest of the digits designate seconds)
4	%C	RTK data availability indicator: “V” means that RTK data are available “N” means that RTK data are unavailable. If this indicator is “N”, the indicator is followed by the checksum.
5	%D	Fixing ambiguity progress, in percentage []
6	%.1F	Estimated probability of fixing ambiguities correctly []
7	%.2F	χ^2 for the position fix []
8	%+.4F	Time shift between the rover and base receiver times [multiplied by the speed of light and presented in meters]
9	%+.4F	Derivative of time offset between the rover and the base [meters/second]
10	%.2F	Root-mean-squared single differenced ionosphere error as estimated by the RTK enginem [meters]
11	%.2F	Corrections age, [seconds]. When RTK works in `extrap` mode, this field contains extrapolation time of data from reference station. When RTK works in `delay` mode, this field is zero.
12	@%2X	Checksum

[AP] Position Covariance Matrix

This message is a text version of the message [SP].

#	Format	Description
1	POSCOV	Message title

#	Format	Description
2	%C	UTC time indicator: "V" means UTC time is valid "N" means UTC time is not valid
3	%6.2F	UTC time of the position fix (the first two digits designate hours, the next two digits designate minutes and the rest of the digits designate seconds)
4	%C	Position covariance matrix indicator: "V" means that Position covariance matrix is available "N" means that Position covariance matrix is not available. If this indicator is "N", the indicator is followed by the checksum.
5	%C	Position computation mode. See Table 4-10 on page 336.
6	%.3E	Cov(1,1)
7	%.3E	Cov(2,2)
8	%.3E	Cov(3,3)
9	%.3E	Cov(4,4)
10	%.3E	Cov(1,2)
11	%.3E	Cov(1,3)
12	%.3E	Cov(1,4)
13	%.3E	Cov(2,3)
14	%.3E	Cov(2,4)
15	%.3E	Cov(3,4)
16	@%2X	Checksum

[AB] Baseline Length

This message contains the length and coordinates of the reference station minus rover baseline.

#	Format	Description
1	BASLIN	Message title
2	%C	UTC time indicator: "V" means UTC time is valid "N" means UTC time is not valid

#	Format	Description
3	%6.2F	UTC time of the position fix (the first two digits designate hours, the next two digits designate minutes and the rest of the digits designate seconds)
4	%C	Baseline length indicator: "V" means that baseline length is available "N" means that baseline length is not available If this indicator is "N", the indicator is followed by the checksum.
5	%C	Position computation mode. See Table 4-10 on page 336.
6	%.4F	X(ref) - X(rover) [meters]
7	%.4F	Y(ref) - Y(rover) [meters]
8	%.4F	Z(ref) - Z(rover) [meters]
9	%.4F	Position SEP [meters]
10	@%2X	Checksum

4.5 Predefined Foreign Messages

4.5.1 Approved NMEA sentences

The NMEA-0183 (National Marine Electronic Association) standard v.2.30 [14] is a specification intended to facilitate interconnection and interchangeability of equipment produced by different manufacturers. The standard defines data transmission specifications, message types and a data exchange protocol between Talker and Listener. It is widely used not only in marine applications but in many other applications too.

The NMEA-0183 standard provides, together with other information, the description of the "approved" sentences. "Approved sentences" are those having predefined formats. Every approved sentence has "talker identifier" and "sentence identifier" and is uniquely characterized by the corresponding (predefined) set of fields. There is a whole variety of devices that may serve as "talkers" in NMEA applications (e.g., marine sounders and weather instruments).

A specific "talker", however, may handle only a particular set of approved messages. For example, a combined GPS/GLONASS receiver utilizes only a limited number of the existing approved sentences (specifically, sentences containing GNSS–related navigational/positioning information).

14. NMEA-0183 Standard For Interfacing Marine Electronic Devices v.3.0. July 1, 2000.

General Format of Approved NMEA Sentences

Each approved NMEA sentence has the following format:

$AACCC,c---c*hh<CR><LF>

where

"$" (HEX 24) – Start of sentence.

AACCC – Address field. The first two characters identify "Talker". The last three characters identify the sentence type.

"," (HEX 2C) – Field delimiter.

c---c – Data sentence block.

"*" (HEX 2A) – Checksum delimiter.

hh – Checksum field. This value is computed by exclusive-OR'ing the eight data bits of each character in the sentence, between, but excluding, "$" and "*". The hexadecimal value of the most significant and least significant four bits of the result are converted into two ASCII characters (0…9,A…F) for transmission. The most significant character is transmitted first.

<CR><LF> (HEX 0D,0A) – sentence terminators. Approved NMEA sentences are allowed to contain the so-called "null fields". Null fields are used when one or more values in the message are unreliable or unavailable. A null field may be delimited by two commas (",,") or by a comma and a multiplication sign "*" (",*") depending on its position in the sentence. JAVAD GNSS receivers support the following "talker identifiers":

"GP" — Global Positioning Systems (GPS)

"GL" — GLONASS

"GN" — Global Navigation Satellite System (GNSS)

Generally speaking, "talker identifier" is supposed to inform Listener whether the positioning information contained in the message is "GPS only", "GLONASS only" or "combined GPS plus GLONASS". In reality, this is not always true: there are sentences whose "talker identifiers" will not indicate the true constellation used in position computation (please see notes to specific sentences listed in the section below).

NMEA-Specific Format Limitations

The NMEA standard forbids the use of character "+" inside approved NMEA sentences. Note that this limitation "overrides" the general format conventions described in "GREIS Format for Text Messages" on page 326.

In other words, in approved NMEA sentences, “+” is omitted before non-negative numbers even if the corresponding field formats contain the plus sign (e.g., %+7.5F).

It should be noted that the NMEA standard, as a rule, does not specify exact mantissa lengths for the sentence fields. The user is free to allocate to every field as many digits as necessary to ensure required accuracy. For example, since JAVAD GNSS receivers provide millimeter-level positioning accuracy in differential modes, receiver geodesic coordinates (latitude - longitude - ellipsoidal height) should have mantissas long enough to enable coordinate presentation with sub-millimeter accuracy. For the format conventions for the following sentences, please see section , “GREIS Format for Text Messages” on page 326.

GGA – Global Positioning System Fix Data

This message comprises time, position and other fix related data for JAVAD GNSS receiver.

#	Format	Description
1	%6.[0-2]F	UTC time of position fix (first two digits designate hours, the next two designate minutes and the rest digits designate seconds)
2	%4.[1-7]F	Latitude in selected datum (first two digits designate degrees and the rest designates minutes of arc)
3	%C	Latitude hemisphere: N – northern, S – southern
4	%5.[1-7]F	Longitude in selected datum (first three digits designate degrees and the rest digits designate minutes of arc)
5	%C	Longitude hemisphere: E – eastern, W – western
6	%1D	GPS quality indicator (see below for details)
7	%2D	Number of satellites used for position computation
8	%.2F	Horizontal dilution of precision (HDOP) [-]
9	%+5.[1-4]F	Altitude above geoid in selected datum [meters]
10	%C	Symbol “M” (denote that altitude is in meters)
11	%+.[1-4]F	Geoidal separation: the difference between the earth ellipsoid and geoid defined by the reference datum [meters]
12	%C	Symbol “M” (denotes that geoidal separation is in meters)
13	%.1F	Age of differential GPS data [seconds]

#	Format	Description
14	%4D	Differential reference station ID (an integer between 0000 and 1023)
15	*%2X\0D\0A	Checksum (see “General Format of Approved NMEA Sentences” on page 345)

Note: The [GGA] message talker identifier uses the following JAVAD GNSS convention: whatever constellation is used for position computation (GPS only, GLONASS only, or GPS plus GLONASS), the talker identifier is always set to “GP”.

If your receiver uses combined GPS+GLONASS data for position computation in RTK or DGPS, “age of differential GPS data” and “differential reference station ID” from [GGA] message will relate to GPS data. On the other hand, if the receiver uses pure GLONASS data when computing the position in RTK or DGPS, the fields “age of differential GPS data” and “differential reference station ID” will relate to GLONASS data.

Generally speaking, it is not recommended to use [GGA] message when operating a full-functionality GPS/GLONASS receiver. Note that [GGA] is mainly intended for pure GPS receivers. For combined receivers, we recommend using [GNS] for [GGA].

GPS quality indicator:

0	Fix not available or invalid
1	GPS SPS Mode (single point mode), fix valid
2	Differential GPS SPS Mode, fix valid
3	GPS PPS Mode (single point mode), fix valid
4	RTK fixed
5	RTK float
6	Estimated (dead reckoning) mode
7	Manual input mode
8	Simulator mode

GLL – Geographic Position – Latitude/Longitude

Current latitude/longitude, time and status of position fix.

#	Format	Description
1	%4.[1-7]F	Latitude in selected datum (first two digits designate degrees and the rest designates minutes of arc)
2	%C	Latitude hemisphere: N – northern, S – southern

#	Format	Description
3	%5.[1-7]F	Longitude in selected datum (first three digits designate degrees and the rest designates minutes of arc)
4	%C	Longitude hemisphere: E – eastern, W – western
5	%6.[0-2]F	UTC time of position (first two digits designate hours, the next two digits designate minutes and the rest designates seconds)
6	%C	Status field (shall be set “V” = Invalid for all values of positioning system mode indicator (see next field) except for “A”= Autonomous, “D”= Differential, “P” = Precise, “R” = RTK with fixed integers and “F” = RTK with floating integers
7	%C	Positioning system mode indicator (see below).
8	*%2X\0D\0A	Checksum (see“General Format of Approved NMEA Sentences” on page 345)

Mode indicator:

A	Autonomous. Satellite system used in non-differential mode in position fix
D	Differential. Satellite system used in differential mode in position fix
E	Estimated (dead reckoning) mode
M	Manual input mode
S	Simulator mode
N	Data not valid

GNS – GNSS Fix Data

This message intended for combined navigation systems (GNSS). It comprises time/position/status fix data.

#	Format	Description
1	%6.[0-2]F	UTC time of position fix (first two digits designate hours, the next two digits designate minutes and the rest designate seconds)
2	%4.[1-7]F	Latitude in selected datum (first two digits designate degrees and the rest designates minutes of arc)
3	%C	Latitude hemisphere: N – northern, S – southern
4	%5.[1-7]F	Longitude in selected datum (first three digits designate degrees and the rest designates minutes of arc)
5	%C	Longitude hemisphere: E – eastern, W – western

#	Format	Description
6	%C%C	Mode indicator (see below): variable length valid character field type with the first two characters currently defined. The first character indicates the use of GPS satellites, the second character indicates the use of GLONASS satellites.
7	%2D	Total number of satellites used for position computation
8	%.2F	Horizontal dilution of precision (HDOP) []
	%+5.[1-4]F	Altitude above geoid in selected datum [meters]
	%+.[1-4]F	Geoidal separation: the difference between the earth ellipsoid and geoid defined by the reference datum [meters]
11	%.1F	Age of differential data [seconds] (see the note below)
12	%4D	Differential reference station ID (this is an integer between 0000 and 1023) (see the note below)
13	*%2X\0D\0A	Checksum (see "General Format of Approved NMEA Sentences" on page 345)

Note: If your JAVAD GNSS receiver runs in "pure GPS" or "pure GLONASS" RTK or DGPS, it outputs one GNS message per position fix. If running in "dual system" RTK or DGPS (i.e., both GPS and GLONASS differential correction data are used simultaneously), the receiver outputs, in accordance with the NMEA standard, a "GNS triplet" for every position fix.

The first message in a GNS triplet plays the most important part carrying the lion's share of information. The other two messages provide some GPS-specific and GLONASS-specific information, specifically: "total number of satellites", "age of differential data" and "differential reference station ID"[15].

The following is the example of a typical GNS triplet:

```
$GNGNS,122310.20,3722.425671,N,12258.856215,W,DD,14,0.9,1005.543,6.5,,*74<CR><LF>
$GPGNS,122310.20,,,,,,7,,,,5.2,23*4D<CR><LF>
$GLGNS,122310.20,,,,,,7,,,,3.0,23*55<CR><LF>
```

Positioning system mode indicator for [GNS] message

N	No fix.
A	Autonomous. Satellite system used in non-differential mode in position fix
D	Differential. Satellite system used in differential mode in position fix
P	Precise. Satellite system used in precision mode. Precision mode is defined as: no deliberate degradation (such as Selective Availability) and higher resolution code (P-code) is used to compute position fix
R	Real time kinematic. Satellite system used in RTK mode with fixed integers

15. Not to mention "UTC time of position fix", which is the same for all three messages in the triple.

F	Float RTK. Satellite system used in real time kinematic mode with floating integers
E	Estimated (dead reckoning) mode
M	Manual input mode
S	Simulator mode

GRS – GNSS Range Residuals

This message contains “range residuals”. These kinds of data are used to support Receiver Autonomous Integrity Monitoring (RAIM).

#	Format	Description
1	%6.[0-2]F	UTC time (the first two digits designate hours, the next two digits designate minutes and the rest designates seconds)
2	%1D	Mode: 0 = residuals were used to calculate the position given in the matching GGA or GNS sentence; 1 = residuals were recomputed after the GGA or GNS position was computed. Currently, the receiver uses only the first mode (Mode = 0).
3	(%+.3F) or (%+.0F)	A sequence of range residuals (in meters). Sequence length depends on the number of satellites used in the position solution. Order must match order of satellite ID numbers in GSA. When GRS is used GSA and GSV are generally required. If the range residual exceeds ±99.9 meters, then the decimal part is discarded, resulting in an integer (-103.7 becomes -103). The maximum value for this field is ±999.
4	*%2X\0D\0A	Checksum (see “General Format of Approved NMEA Sentences” on page 345)

Note: The NMEA standard states the following:

- If either GPS or GLONASS is used for position computation, the talker ID is set to “GP” or “GL”, respectively.

- If GPS and GLONASS are used together, the receiver will generate two GRS messages at one time. The first of these messages will describe the GPS range residuals whereas the other will describe the GLONASS range residuals. Either message will have the same talker ID, “GN”, which indicates that the range residuals actually relate to GNSS.

GSA – GNSS DOP and Active Satellites

This message describes GNSS receiver operating mode, satellites used in the position solution reported by the GGA or GNS sentence, and DOP values.

#	Format	Description
1	%C	Mode: M = Manual, forced to operate in 2D or 3D mode, A = Automatic, allowed to automatically switch 2D/3D
2	%D	Mode: 1 = Fix not available, 2 = 2D, 3 = 3D
3	%.2D	A sequence of satellite ID numbers. Sequence length is variable (depends on the amount of satellites used in solution). For more details on satellite ID numbers, see below.
4	%.2F	Position dilution of precision (PDOP)
5	%.2F	Horizontal dilution of precision (HDOP)
6	%.2F	Vertical dilution of precision (VDOP)
7	*%2X\0D\0A	Checksum (see “General Format of Approved NMEA Sentences” on page 345)

The NMEA standard states the following:

- If either GPS or GLONASS is used for position computation, the talker ID is set to “GP” or “GL”, respectively.
- If GPS and GLONASS are used together, two GSA messages are generated at one time. The first and second messages relate to the GPS and GLONASS satellites, respectively. Both the messages, however, will have the same talker ID, “GN”, and the same DOP values (the latter are actually computed for the combined constellation). The talker ID “GN” indicates that this pair of messages relate to the one and the same GNSS solution.

Table 4-11. Satellite ID Numbers

NMEA Satellite ID Numbers	System Numbers
1…32	GPS PRN numbers 1…32
65…88	GLONASS slot numbers [1…24]

GST – GNSS Pseudorange Error Statistics

This message is used to support Receiver Autonomous Integrity Monitoring (RAIM).

#	Format	Meaning
1	%6.[0-2]F	UTC time of position (the first two digits designate hours, the next two digits designate minutes and the rest designate seconds)
2	%.3F	Estimated standard deviation of the range input's error. “SV Range input”, which is used in the navigation process, includes this satellite’s pseudo-range and the corresponding DGNSS correction [meters]
3	%.3F	Semi-major axis of error ellipse [meters]
4	%.3F	Semi-minor axis of error ellipse [meters]
5	%.3F	Orientation of semi-major axis of error ellipse [degrees from true north]
6	%.3F	RMS latitude error [meters]
7	%.3F	RMS longitude error [meters]
8	%.3F	RMS altitude error [meters]
9	*%2X\0D\0A	Checksum (see “General Format of Approved NMEA Sentences” on page 345)

Note: In case the solution computed by the receiver is not RTK fixed or RTK float, fields 3, 4 and 5 are null fields.

GSV – GNSS Satellites in View

Number of satellites in view, satellite ID numbers, elevation, azimuth and SNR value.

#	Format	Meaning
1	%1D	Total number of messages, 1 to 3
2	%1D	Message number, 1 to 3
3	%2D	Total number of satellites in view
4	(%2D,%2D, %3D,%2D)	Satellite ID number (see GSA for ID numbers), elevation in degrees, azimuth in degrees and C/A signal-to-noise ratio (SNR) in dB*Hz
5	*%2X\0D\0A	Checksum (see “General Format of Approved NMEA Sentences” on page 345)

Note: A variable number of "Satellite ID-Elevation-Azimuth-SNR" sets are allowed (up to a maximum of four sets per message).

In case the number of visible SVs exceeds 4, multiple messages are transmitted. The first field specifies the total number of messages (minimum value 1) whereas the second field identifies the order of this message (i.e., message number), minimum value 1.

Messages for GPS and GLONASS are generated separately: GPS messages will have Talker ID "GP" and GLONASS messages - Talker ID "GL".

Note that the number of SVs in view can sometimes exceed 12.

Since the NMEA standard allows only three messages to be generated per epoch, a maximum of 12 SVs can be output at a given epoch. Therefore, if the total number of satellites in sight exceeds 12, there may be visible satellites not included in any of the GSV messages related to the given epoch. The following is an example "GSV" message (all of the lines correspond to the same epoch):

```
⇐ $GPGSV,3,1,10…<CR><LF>
⇐ $GPGSV,3,2,10…<CR><LF>
⇐ $GPGSV,3,3,10…<CR><LF>
⇐ $GLGSV,2,1,7…<CR><LF>
⇐ $GLGSV,2,2,7…<CR><LF>
```

RMC – Recommended Minimum Specific

GNSS Data Time, date, position, course and speed data provided by a GNSS navigation receiver.

#	Format	Description
1	%6.[0-2]F	UTC time of position fix (first two digits designate be hours, the next two designate minutes and the rest digits designate seconds)
2	%C	Status: A – Data valid, V – Navigation receiver warning
3	%4.[1-7]F	Latitude in selected datum (first two digits designate degrees and the rest designates minutes of arc)
4	%C	Latitude hemisphere: N – northern, S – southern
5	%5.[1-7]F	Longitude in selected datum (first three digits designate degrees and the rest digits designate minutes of arc)
6	%C	Longitude hemisphere: E – eastern, W – western
7	%.[1-4]F	Speed over ground (horizontal speed) [knots]
8	%3.[1-3]F	Course over ground (true course) [degrees]

#	Format	Description
9	%S	Date: DDMMYY
10	%3.[1-3]F	Magnetic variation [degrees]
11	%C	Magnetic variation direction: E – eastern, W – western
12	%C	Mode indicator (see Positioning system mode indicator in GLL message above)
13	*%2X\0D\0A	Checksum (see “General Format of Approved NMEA Sentences” on page 345)

HDT – Heading, True

#	Format	Description
1	%.3F	True Heading in degrees
2	%C	Symbol “T” indicates true heading
3	*%2X\0D\0A	Checksum (see “General Format of Approved NMEA Sentences” on page 345)

VTG – Course Over Ground and Ground Speed

The actual course and speed relative to the ground.

#	Format	Description
1	%3.[1-3]F	True course [degrees]
2	%C	Symbol “T” indicates True course
3	%3.[1-3]F	Magnetic course [degrees]
4	%C	Symbol “M” indicates Magnetic course
5	%.[1-4]F	Horizontal speed [knots]
6	%C	Symbol “N” indicates that horizontal speed is given in knots
7	%.[1-4]F	Horizontal speed [km/h]
8	%C	Symbol “K” indicates that horizontal speed is given in km/h
9	%C	Mode indicator (see “GGA – Global Positioning System Fix Data” on page 346)

#	Format	Description
10	*%2X\0D\0A	Checksum (see “General Format of Approved NMEA Sentences” on page 345)

ROT – Rate of Turn

#	Format	Description
1	%.3F	Rate of turn, degrees/minute. If negative, bow turns to port else bow turns to starboard 2
2	%C	A = data valid, V = data invalid
3	*%2X\0D\0A	Checksum (see “General Format of Approved NMEA Sentences” on page 345)

Note: This message is available if heading mode is turned on (i.e., `/par/pos/pd/hd/mode` is set to `on`).

ZDA – UTC Time and Date

#	Format	Description
1		%6.[0-2]F UTC time (first two digits designate hours, next two digits designate minutes and the rest designates seconds)
2	%2D	Day (varies between 01…31)
3	%2D	Month (varies between 01…12) 4 6
4	%4D	Year
5	%2D	Local zone hours (varies from -13 to +13)
6	%2D	Local zone minutes (varies from 00 to 59)
7	*%2X\0D\0A	Checksum (see “General Format of Approved NMEA Sentences” on page 345)

Note: Local time zone is the magnitude of hours plus the magnitude of minutes added, with the sign of local zone hours, to local time to obtain UTC.

To specify values of local zone hours and local zone minutes, use the command `set,/par/pos/ltz,{H,M}` (see“Positioning Parameters” on page 83).

[GMP] - GNSS Map Projection Fix Data

This message contains fix data for single or combined navigation systems (GNSS) in grid (or local) coordinates expressed in the given map projection.

#	Format	Description
1	%6.[0-2]F	UTC time of position fix (first two digits designate hours, the next two digits designate minutes and the rest digits designate seconds).
2	%S	Map projection identification.
3	%S	Map zone.
4	%.[1-4]F	Y (Northern) component of grid (or local) cored-nates in meters.
5	%.[1-4]F	X (Eastern) component of grid (or local) coordinates in meters.
6	%C%C	Mode indicator (see GNS message): variable length valid character field type with the first two characters currently defined. The first character indicates the use of GPS satellites, the second character indicates the use of GLONASS satellites.
7	%2D	Total number of satellites used for position computation.
8	%.2F	Horizontal dilution of precision (HDOP) [-]
9	%+5.[1-4]F	Altitude above geoid in the selected datum [meters]
10	%+.[1-4]F	Geoidal separation: the difference between the earth ellipsoid and geoid defined by the reference datum [meters]
11	%.1F	Age of differential data [seconds]. See the note below.
12	%4D	Differential reference station ID (this is an integer between 0000 and 1023). See the note below.
13	*%2X\0D\0A	Checksum

Note: All the notes applicable to the NMEA GNS sentence are valid for GMP message as well.

4.5.2 JNS Proprietary NMEA Sentences

All JNS proprietary NMEA sentences should have the following format:

$PTPSR,MsgID,c---c*hh<CR><LF>

ATT – Attitude Parameters

#	Format	Description
1	%C	UTC time indicator: "V" means that UTC time is valid "N" means that UTC time is not valid
2	%6.2F	UTC time of position (the first two digits designate hours, the next two digits designate minutes and the rest of the digits designate seconds)
3	%.3F	True Heading in degrees
4	%.3F	Pitch in degrees
5	%C	Base position type ('N' – not available, 'A' – autonomous, 'D' – code differential, 'F' – "float", 'R' – fixed). This field can be useful provided JPS messages are used for broadcasting RTK data
6	*%2X\0D\0A	Checksum (see , "General Format of Approved NMEA Sentences" on page 345)

Note: To enable [ATT] message, use the message name `/msg/nmea/P_ATT`

4.5.3 RTCM 2.x Messages

Introduction to RTCM 2.x Messages

RTCM (Radio Technical Commission For Maritime Services) SC-104 (Special Committee 104) has developed a standard for differential GNSS (Global Navigation Satellite Systems) service[16]. The standard formulates recommendations in the following areas:

1. Data message and format – The message elements that make up the corrections, the status messages, the station parameters and ancillary data are defined in some details.
2. User interface – A standard interface is defined which enables a receiver to be used in concert with a variety of different data links.

The formats both GPS/GLONASS RTK data and GPS/GLONASS differential corrections are defined as well as the formats of the messages, which includes, e.g., reference station parameters, special message etc.

16. See for details: RTCM recommended standards for differential GNSS (Global Navigation Satellite System) service, version 2.3, August 20, 2001. (RTCM PAPER 136-2001/SC104-STD)

This standard is used, for example, for broadcasting the differential corrections by the radio-beacon based differential services located near coastal waters all over the world.

Every RTCM message consists of a variable number of 30-bit words (the reader will notice some resemblance to the structure of the GPS Navigation Message). The first two words in any RTCM message serve as the message header. The header comprises the following data fields: preamble, message type, reference station ID, modified Z-count, sequence number, length of frame and station health. The contents of the other words will depend on the type of the message (see 9 for more details).

Supported RTCM 2.x Messages

Table below contains a list of RTCM messages currently supported by JAVAD GNSS receivers. The column “type” indicates the message type number as specified in the RTCM standard.

The second column shows the message's GREIS-specific identifier (or simply “ID”). When the user enables a particular RTCM message with an appropriate command, he/she specifies the message “ID”, not the message “type”.

The third column, “title”, describes the message contents. 203The fourth column, “period”, specifies the default periods of the corresponding RTCM messages.

The last column explains how to calculate an RTCM message's length in 30-bit words.

Note that according to the RTCM standard it takes the receiver five bytes52 to transmit a 30-bit word. This is because the two MSB in each of these five bytes are “reserved” (more precisely, set to some pre-defined values). Thus the following formula to compute the actual length of an RTCM message [in bits]:

$$[\text{Length in bits}] = [\text{Length in 30-bit words}] \times 5 \times 8$$

Note: N designates the total number of satellites, and S designates the length (in characters) of the user-specified text

Type	ID	Title	Period [s]	Length in 30-bit words
1	1	Differential GPS corrections	1	2+N*2-N/3
3	3	GPS reference station parameters	10	2+4
6	6	GPS null frame	30	2
9	9	GPS partial correction set	1	2+N*2-N/3
15	15	Ionospheric delay message	60	2+N*2-N/2

Type	ID	Title	Period [s]	Length in 30-bit words
16	16	GPS special message	30	2+1+(S-1)/3
18	18	RTK uncorrected carrier phases	1	3+N*2 for C/A or P L1 data 2*(3+N*2) for L1+L2 data
19	19	RTK uncorrected pseudoranges	1	3+N*2 for C/A or P L1 data 2*(3+N*2) for L1+L2 data
20	20	RTK carrier phase corrections	1	3+N*2 for C/A or P L1 data 2*(3+N*2) for L1+L2 data
21	21	RTK/High accuracy pseudorange corrections	1	3+N*2 for C/A or P L1 data 2*(3+N*2) for L1+L2 data
22	22	Extended reference station parameters	10	(2+1) to (2+3)
23	23	Antenna Type Definition Record	10	2+1+[Nant+Nser] Nant - the number of characters used for antenna descriptor; Nser – the number of characters used for serial number.
24	24	Reference Station Antenna Reference Point Parameter, which provides the exact location of the reference station and the antenna height as the distance to the Antenna Reference Point (ARP).	10	5 (if the antenna height is not included in the message) or 6 (if the antenna height is included in the message).
31	31	Differential GLONASS corrections	1	2+N*2-N/3
32	32	GLONASS reference station parameters	10	2+4
34	34	GLONASS partial correction set	1	2+N*2-N/3
34	65	GLONASS null frame (34^{th} message with N=0)	30	2
36	36	GLONASS special message	30	2+1+(S-1)/3

Note: Unless you want to provide backward compatibility with earlier versions of the RTCM format, JAVAD GNSS recommends that you use messages 23 and 24 instead of messages 3 and 22. To retain backward compatibility with previous versions of RTCM format, it is recommended to transmit both pairs of the messages in the same RTK data stream.

This table indicates how many 30-bit words each specific RTCM message takes. For information on the total amount of data (in bytes) transmitted by the reference station in RTK or DGPS, see *“GREIS User’s Manual”* available from *http://www.javad.com*.

4.5.4 RTCM 3.0 Messages

RTCM (Radio Technical Commission For Maritime Services) SC-104 (Special Committee 104) has developed a standard for differential GNSS (Global Navigation Satellite Systems) service[17].

The following RTCM 3.0 messages are supported:

- 1003 – GPS Basic RTK, L1 and L2
- 1004 – GPS Extended RTK, L1 and L2
- 1005 – Stationary Antenna Reference Point without Height Information
- 1006 – Stationary Antenna Reference Point with Height Information
- 1007 – Antenna Descriptor
- 1008 – Antenna Descriptor and Serial Number
- 1011 – GLONASS Basic RTK, L1 and L2
- 1012 – GLONASS Extended RTK, L1 and L2
- 1019 – GPS ephemeris (include GPS ephemeris in accordance with current RTCM documents)
- 1020 – GLONASS ephemeris (include GLONASS ephemeris in accordance with current RTCM documents).
- 4090t – Proprietary text message. This message provides a possibility to transmit a text message from the base to the rover receiver. It is a proprietary message, thus the receivers of other developers cannot use this message.

Example: To enable the output of message 1004 on serial port C with the period 1 second, issue the following command:

⇒ `em,/dev/ser/c,/msg/rtcm3/1004:1.0`

Example: To enable the output of messages 1019 and 1020 through, e.g., serial port C, use the following command:

⇒ `em,/dev/ser/c,/msg/rtcm3/{1019,1020}`

Example: The command that enables the output of this message is the following:

⇒ `em,/dev/ser/c,/msg/rtcm3/4091t`

17. See for details: RTCM recommended standards for differential GNSS (Global Navigation Satellite System) service, version 3.0, February, 10, 2004. (RTCM PAPER 30-2004/SC104-STD)

4.5.5 CMR Messages

Introduction to CMR Messages

The Compact Measurement Record (CMR) format was developed by Trimble Navigation Limited and now is approved for public use. The format is suitable for communication links that have a minimum of 2400 baud throughput (assuming that only GPS data is used). It provides significant advantages over RTCM messages (the latter are at least twice as long as compared against their CMR counterparts). It should be noted however that the original version of the CMR format does not allow for GLONASS. Therefore the original format definition needs to be expanded to include GLONASS.

For a detailed description of the CMR format see ftp://ftp.trimble.com/pub/survey/cmr.

Supported CMR Messages

Table below lists CMR messages currently supported by JAVAD GNSS receivers.

The column “type” indicates the message type as specified in the CMR standard except the message type var(3). GREIS defined this type to overcome some limitations existing in the CMR protocol.

The second column shows what GREIS specific identifiers (IDs) are assigned to different CMR messages. When enabling a CMR message with an appropriate GREIS command, the user specifies its “ID”, not “type”.

The third column, “title”, describes what kind of data each CMR message contains.

The fourth column, “period”, defines default periods for CMR messages.

The last column explains how to calculate the length of a CMR message (in bytes).

The following notations are used:

`N` – designates the total number of satellites.

`S` – is the length (in characters) of Long Station ID.

`FREQ` – takes 1 and 2 for single- and dual-frequency measurements, respectively.

`Var` – indicates that the type is subject to change

Type	ID	Title	Period (seconds)	Length in 30-bit words
0	0	GPS measurements	1	6 + N * (8 + (FREQ-1) * 7)

Type	ID	Title	Period (seconds)	Length in 30-bit words
1	1	Reference station coordinates	10	31
2	2	Reference station description	10	31+S
var(3)	10	GLONASS measurements (GREIS proprietary message)	1	6 + N * (8 + (FREQ-1) * 7)
+	9[1]	Scrolling Station Information	1	16

1. This message has been developed for reducing the peak throughput of the CMR format. It is used for CMR message types 1 and 2. Data from this message type is transmitted by “frames”. Each frame is 16 bytes in size (7-byte message body plus 9 auxiliary bytes), and is transmitted together with CMR Message Types 0 and 10, which allows considerable reduction of the data link peak load. For more details about this message, see ftp://ftp.trimble.com/pub/survey/bin/uconf97.exe

4.5.6 BINEX Messages

BINEX, for “BINary EXchange”, is an operational binary format standard for GPS/GLONASS/SBAS research purposes. It has been designed to allow encapsulation of all (or most) of the information currently allowed for in RINEX OBS, GPS RINEX NAV, GLONASS RINEX NAV, RINEX MET, IONEX, SP3, SINEX, and so on, plus other GNSS-related data and metadata as encountered, including next-generation GNSS.

For a detailed information on BINEX specifications, refer to UNAVCO's Web site at *http://www.binex.unavco.org*.

This group comprises the following BINEX messages:

`/msg/binex/00_00` – BINEX record `0x00-00`

`/msg/binex/01_01` – BINEX record `0x01-01`

`/msg/binex/7D_00` – BINEX record `0x7D-00`

`/msg/binex/7E_00` – BINEX record `0x7E-00`

`/msg/binex/7F_02` – BINEX record `0x7F-02`

`/msg/binex/7F_03` – BINEX record `0x7F-03`

`/msg/binex/7F_04` – BINEX record `0x7F-04`

For the BINEX record `0x00-00`, the following fields are supported:

```
0x04 0x0f 0x17 0x19 0x1a 0x1b 0x1d 0x1f
```

It's possible to turn on/off the output of each of the above fields using the `/par/binex/00_00` parameter.

The values for fields `0x04` and `0x0f` could be specified using parameters `/par/binex/site` and `/par/binex/data_id`, respectively.

Meteorological data for BINEX record `0x7E-00` could be obtained by connecting MET3-compatible sensor to a receiver port, setting the `imode` of the port to `jps` and enabling output of `/msg/misc/MET3` to this port.

Example: Program receiver to get data from MET3 sensor working at 9600 baud and connected to the serial port B. The MET3 data are requested every 60 seconds with offset -2 seconds (phase =86400-2) so that the data are ready by the time BINEX record `0x7F-02` is output. Then request output of BINEX record `0x7F-02` every 60 seconds to the serial port A:

```
⇒ set,/par/dev/ser/b/rate,9600
⇒ set,/par/dev/ser/b/imode,jps
⇒ em,/dev/ser/b,/msg/misc/MET3:{60,86398}
⇒ em,/dev/ser/a,/msg/binex/7E_00:60
```

●

APPENDICES

A.1 Computing Checksums

For messages, the checksum is computed starting with the first byte of the message identifier and ending with the byte immediately preceding the checksum field, inclusive.

For commands, the checksum is computed starting with the command's first non-blank byte and ending with the character '@', inclusive.

A.1.1 Computing 8-bit Checksum

Provided 'count' bytes of data are put into a buffer 'src', the 8-bit checksum could be computed according to the following algorithm:

```
typedef unsigned char u1;

enum {
  bits   = 8,
  lShift = 2,
  rShift = bits - lShift
};

#define ROT_LEFT(val) ((val << lShift) | (val >> rShift))

u1 cs(u1 const* src, int count)
{
  u1 res = 0;
  while(count--)
    res = ROT_LEFT(res) ^ *src++;
  return ROT_LEFT(res);
}
```

A.1.2 Computing CRC16

Provided 'count' bytes of data are put into a buffer 'src', the CRC16 checksum could be computed according to the following algorithm:

```
typedef unsigned short Crc16;

enum {
  WIDTH = 16,     // Width of poly
  POLY = 0x1021, // Poly. Bit #16 is set and hidden
  BYTE_BITS = 8, // Number of bits in byte
```

```
  TABLE_SIZE = 1 << BYTE_BITS, // Size of table
  MSB_MASK = 1 << (WIDTH - 1)  // Mask for high order bit in a word
};

// Table (generated by 'crc16init()'.
static Crc16 table[TABLE_SIZE];

// Initializes the table. Should be called once before the first
// call to 'crc16()'
void crc16init(void)
{
  Crc16 i;
  for(i = 0; i < TABLE_SIZE; ++i) {
    Crc16 val = i << (WIDTH - BYTE_BITS);
    int j;
    for(j = 0; j < BYTE_BITS; ++j)
      val = (val << 1) ^ ((val & MSB_MASK) ? POLY : 0);
    table[i] = val;
  }
}

// Calculates CRC16 of 'cnt' bytes from 'src' and returns result.
// Initial value of CRC16 is supplied by caller in 'crc'.
Crc16 crc16(Crc16 crc, void const* src, int cnt)
{
  unsigned char const* s = (unsigned char const*)src;
  while(cnt--) {
    crc = (crc << BYTE_BITS)
      ^ table[(crc >> (WIDTH - BYTE_BITS)) ^ *s++];
  }
  return crc;
}
```

When the crc16() function is used to calculate checksum of a receiver message, the initial value of CRC16 should be set to zero.

A.2 Data Transfer Protocol

The GREIS data transfer protocol (DTP) is primarily designed for downloading measurement files from JAVAD GNSS receivers to a host computer and to upload new firmware to the JAVAD GNSS receivers. The process of downloading or uploading should be initiated by corresponding GREIS command(s) sent to the JAVAD GNSS receiver. After a transfer is initiated, parties should use the protocol described here.

In this section the terms “transmitter” and “receiver” are used to denote the data source and destination, respectively, of the data transfer protocol. We will call the JAVAD GNSS board “JAVAD GNSS receiver” to distinguish it from the receiving end of the protocol.

A.2.1 Protocol Description

The protocol is a fixed-size block protocol with checksum and the ability to re-send a block multiple times should a transmission error occur.

For the purposes of transmission, the stream of bytes to be transferred is divided into stream of blocks of fixed size, except the size of the last block that could be smaller. The size of data blocks is negotiated between parties in advance, before the protocol starts. Each block of data is assigned its number starting from zero for the first block. The blocks of data are then transferred from transmitter to receiver using the following formats and procedures.

The format of a single block of transmission could be represented by the following C structure, where all multi-byte fields are sent in the "least significant byte first" (LSB) order:

```
struct Block {
  u1 type;         // Block type:
                   //  ORDINAL (SOB, 0x02)
                   //  LAST (EOT, 0x04)
                   //  ABORT (#, 0x23)
  // The following fields do not exist for block of type ABORT.
  union {
    u2 number;     // Block number for block of type
                   // ORDINAL (0 - based).
    i2 count;      // Number of bytes of data in the
                   // block of type LAST. Value -1
                   // indicates transfer error. In this
                   // case 'data' field of the block
                   // contains a zero-terminated string
                   // describing the error type.
  };
  u1 data[size];   // Data block. In the block of type LAST
                   // only the first 'count' bytes are
                   // valid. The rest of the bytes are filled
                   // with zeroes.
  u2 cs;           // CRC16 checksum of bytes starting from
                   // the 'type' field up to, but not
                   // including, 'cs' field. It is calculated
                   // through all blocks starting from block 0.
  u1 eob;          // End Of Block marker (EOB, 0x03).
};
```

The transmitter sends blocks of this format in response to the receiver's requests. Requests are single byte values. There are three types of requests:

1. NACK, negative acknowledge, code 0x15. Request to re-send the current block.
2. ACK, positive acknowledge, code 0x06. Request to send the next block.
3. ABORT, abort transfer. One of the characters in the range [!–/] (ASCII codes [0x21–0x2F]). Particular value sent could be used to denote the kind of error occurred.

Any other value received when request is expected is ignored by the transmitter.

The NACK request is sent in the following cases:

- After the transfer protocol is initiated, to ask the transmitter to send the first block (block number zero).
- When receiving error occurs[1], to ask the transmitter to re-send the last block.

The ACK request is sent after the block with matching block number is successfully received and passed the test for data integrity, to ask the transmitter to send the next block.

The ABORT request could be sent by the receiver instead of ACK or NACK request to terminate the transfer protocol.

To deal with the possibility of losing synchronization between the transmitter and receiver, receiver should keep track of the expected block number. Only if actual block number of successfully received block is equal to the expected block number, receiver should send the ACK request. If successfully received block has the number that is less than those expected, receiver should silently ignore the block as it's just another copy of previously received block. If received block number is greater than those expected[2], receiver should stop the protocol by sending the ABORT request due to unrecoverable loss of synchronization.

The protocol can be terminated by the following events:

1. Transmitter sends a block of type LAST with a positive "count" field. This is a normal end of transfer. In this case the last "count" bytes of data are in the "data" field of the block. The rest of the bytes of the "data" field are filled with zeroes.
2. Transmitter sends a block of type LAST with a "count" field equal to "-1" (minus one). This means that an error on the transmitter end has occurred. The "data" field of the block contains a zero-terminated ASCII string describing the error.
3. Transmitter sends a block of type ABORT (i.e. sends "#" instead of SOB or EOT). This means that the receiver should immediately terminate its operation.
4. Receiver sends the ABORT request instead of ACK or NACK request. This means that an error on the receiver end has occurred. In this case the particular value sent denotes the error code.

1. Including the case when no data is received in reasonable time after previous ACK or NACK request.
2. Though unlikely, this could still happen should NACK sent by the receiver be received as ACK by the transmitter due to transmission error.

A.2.2 Checksum Calculation

The CRC16 checksum is calculated from the bytes of blocks starting from the field “type” up to but not including the field “cs”. The initial value of “cs” is set to zero (0) at the beginning of the transfer session. Each additional block uses the initial value for “cs” obtained from the previous successfully sent and received block.

Assuming the received block is stored into the buffer named “block”, the checksum validation could be done as follows (see “Computing CRC16” on page 365 for implementation of the crc16() function):

```
unsigned char block[1 + 2 + 512 + 2 + 1];
Crc16 crc = 0;
…
// NOTE: crc isn't reset to 0 at each block.
crc = crc16(crc, block, 1 + 2 + 512);
if(crc == crcReceived)
  // Checksum is correct
else
  // Checksum is wrong
```

To achieve this result, the transmitter calculates the “cs” field as follows:

```
unsigned char block[1 + 2 + 512 + 2 + 1];
Crc16 crc = 0;
…
/* NOTE: crc isn't reset to 0 at each block. */
crc = crc16(crc, block, 1 + 2 + 512);
/* Resulting 'crc' is output by transmitter LSB first. */
```

A.3 Compensating for Phase Rollovers

Carrier phases from receiver messages [pc], [p1], and [p2] will have discontinuities due to the periodic rollovers of the 32-bit word used to represent the carrier phase in these messages. Such rollovers don't occur very frequently (approximately once every 15 minutes or more). If you wish to use these messages and want to remove this kind of discontinuities from the carrier phases, use the following correction technique.

The C function *phase()* below recovers full phase:

```
typedef unisgned long u32;

/*
Return full carrier phase in cycles provided ‘prev’ contains short carrier
phase from previous epoch, ‘cur’ contains short carrier phase at current
epoch, and ‘rollovers’ points to the variable containing number of detected
rollovers so far.
Updates the number of detected rollovers.
*/
```

```
double phase(u32 curr, u32 prev, u32* rollovers)
{
  if(curr < (1 << 30) && prev > (1 << 30) * 3)
    ++*rollovers;
  if(curr > (1 << 30) * 3 && prev < (1 << 30))
    --*rollovers;
  return ((1 << 16) * (double)(*rollovers) * (1 << 16) + curr) / 1024.;
}
```

Pseudo-code for a program utilizing the above function is:

```
u32 rollovers = 0;
u32 curr;
u32 prev;

SEEK_TO_THE_FIRST_EPOCH;
prev = GET_SHORT_PHASE_FORM_CURRENT_EPOCH;
loop {
  curr = GET_SHORT_PHASE_FORM_CURRENT_EPOCH;
  truePhase = phase(curr, prev, &rollovers);
  prev = curr;
  OUTPUT_PHASE(truePhase);
  SEEK_TO_THE_NEX_EPOCH;
}
```

A.4 Obsolete Receiver Objects

In this section you will find a list of receiver objects that are considered obsolete, along with recommended substitutions. Some of obsolete objects listed in the table below are not supported anymore. Others are still supported for backward compatibility but are subject for removal without notice. Obsolete objects that are still supported are only recognized when explicitly adressed in the GREIS commands, i.e., they are not output by `print` and `list` commands when containing group of objects is being output.

Table A-1. Obsolete Receiver Objects

Obsolete Object	Use Instead
/par/version/main	/par/rcv/ver/main
/par/rcvid	/par/rcv/id
/par/raw/elm	/par/out/elm/cur/file/a
/par/button/period	/par/log/sc/period
/par/rtcm/rover/maxage	/par/pos/cd/maxage
/par/rtcm/mode	/par/base/mode/rtcm /par/rover/mode/rtcm
/par/pos/meas	/par/pos/sp/meas
/par/pos/iono	/par/pos/sp/iono

Table A-1. Obsolete Receiver Objects

Obsolete Object	Use Instead
/par/pos/tropo	/par/pos/sp/tropo
/par/hd/mode	/par/pos/pd/hd/mode
/par/hd/uselen	/par/pos/pd/hd/uselen
/par/hd/len/0	/par/pos/pd/hd/len/0
/par/opts/all/NAME	/par/opts/NAME
/par/opts/cur/NAME	/par/opts/NAME/cur
/par/hist/out	/par/pos/pd/hist/out /par/pos/pd/hist/num /par/pos/pd/hist/bad
/par/out/elm/cur/log	/par/out/elm/cur/file/a
/par/out/minsvs/cur/log	/par/out/minsvs/cur/file/a
/par/out/epochs/cur/log	/par/out/epochs/cur/file/a
/cur/log	/cur/file/a
/par/out/cur/log	/par/out/cur/file/a
/par/log/sc/period	/par/log/a/sc/period
/par/cmd/create/prefix	/par/cmd/create/pre/a
/par/rover/base/ant/got/offs	/par/rover/base/ant/got/m_offs
/log/&[name,size,time]	/log (i.e., print,/log:on)

1731 Technology Drive, San Jose, CA 95110 USA
Phone: +1(408)573-8100
Fax: +1(408)573-9100
www.javad.com

GREIS (GNSS Receiver External Interface Specification)